UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Quadcopter simulation using a hardware-in-the-loop technique

Trabajo de Final de Grado

Grado en Ingeniería Informática-Ingeniería de Computadores

Alex Archilla Sanchez
Director: Antoni Grau Saldes
Codirector: Edmundo Guerra Paradas

Q2 2018-2019

Agradecimientos

Me gustaría utilizar este espacio para agradecer a todas aquellas personas que de forma directa o indirecta, han contribuido en el desarrollo de este proyecto.

En primer lugar, me gustaría agradecer al director Antoni Grau y al subdirector Edmundo Guerra por sus consejos, su entrega, su ayuda y todo el tiempo que han dedicado en este proyecto.

También me gustaría agradecer la paciencia de todos y cada uno de los integrantes del departamento por ayudarme a localizar el material y por su paciencia a la hora de aguantar los ruidos generados por la controladora.

Finalmente, agradecer a mis padres, abuelos y a mi hermana todo el apoyo dado durante todo mi transcurso universitario.

Resumen

El mercado de los drones ha evolucionado mucho estos últimos años y cada vez se encuentra más presente en diferentes sectores. Por esto mismo, se tiene que asegurar la fiabilidad y la seguridad de estas aeronaves.

El objetivo de este trabajo de final de grado es la creación de un entorno de simulación para multicopteros. La plataforma de simulación se basa en Gazebo junto a ROS como middleware entre el simulador y el dron. Para validar el control de navegación del drone, se utilizará la placa Pixhawk con su software de control de vuelo de código abierto PX4 mediante el uso de la técnica de simulación hardware in the loop.

Resum

El mercat de los drons ha evolucionat molt durant els últims anys i cada cop la trobem més present en diferents sectors. Per això mateix, s'ha d'assegurar la fiabilitat i la seguretat d'aquestes aeronaus.

L'objectiu d'aquest treball de final de grau és la creació d'un entorn de simulació per multicopters. La plataforma de simulació es basa en Gazebo juntament amb ROS com a middleware entre el simulador i el dron. Per validar el control de navegació del dron, s'utilitzarà la placa Pixhawk juntament amb el software de control de vol de codi obert PX4 mitjançant l'ús de la tècnica de simulació hardware in the loop.

Abstract

The drone market has evolved a lot in recent years and we are finding it more and more present in different sectors. For this reason, the reliability and safety of these aircraft must be ensured.

The purpose of this end-of-grade paper is to create a simulation environment for multicopters. The simulation is based on Gazebo together with ROS as middleware between the simulation and the drone. To validate the drone navigation control, a real board (Pixhack) with its open source flight control software PX4 will be used, creating a hardware-in-the-loop approach.

Índice

Índice de figuras

Índice de tablas

Lista de Acrónimos

UAV	Unmanned Aerial Vehicle. Vehículo aéreo no tripulado.
UAS	Unmanned Aerial System. Sistema aéreo no tripulado.
HIL/HITL	Hardware in the Loop. Técnica de simulación de sistemas.
SIL/SITL	Software in the Loop. Técnica de simulación de sistemas.
GGC	QGroundControl. Software de estación de drones en tierra.
BSD	Berkeley Software Distribution. Tipo de licencia de software libre.
RTOS	Real-Time Operating System. Sistema operativo de tiempo real.
IMU	Inertial Measurement Unit. Unidad de medición inercial.
UDP	User Datagrama Protocol. Protocolo de datagramas de usuario.
EDAR	Estación Depuradora de Aguas Residuales
MAVLink	Micro Air Vehicle Link. Protocolo estándar de comunicaciones de drones.
Camara FPV	Camara First Person View. Cámara de vista en primera persona.
ODE	Open Dynamics Engine. Motor de dinámicas de Gazebo.
GPS	Global Positioning System. Sistema de posicionamiento global.
SDF	Simulation Description Format. Descripción del formato de simulación. Lenguaje de modelado de robots de Gazebo.
SURF	Speeded-Up Robust Features. Algoritmo de visión por computador.
SLAM	Simultaneous Localization and Mapping. Localización y modelado simultáneos.
URDF	Universal Robot Description Format. Formato universal de descripción de robot.

1. Introducción

1.1 Contexto

Los UAV o comúnmente drones, es cualquier aeronave cuya principal característica es que vuela sin tripulación, siendo capaz de volar de forma autónoma y/o controlada. Este término acuña una gran variedad de vehículos, ya que hace referencia a cualquier vehículo aéreo no tripulado, ya sea de forma autónoma o mediante control remoto. Adicionalmente, con este término, solo nos estamos refiriendo de forma exclusiva a la aeronave. Por eso se introdujo el término UAS, el cual además de la aeronave, incluye también la estación de drones y su sistema de comunicación.

Como se mencionó anteriormente, hay una gran variedad de UAV. Estos aparatos de pueden dividir en 3 grandes grupos: Drones de alas fijas, drones de alas rotatoria y multirotor. En este proyecto nos centraremos principalmente en el último grupo, específicamente en los quadcopters, ya que son el tipo de dron más común más usado en el departamento.

Durante los últimos años, ha habido un gran avance tecnológico en este campo, y están apareciendo muchas líneas de investigación y muchos posibles usos para esta tecnología. Sin embargo, este proyecto se basa en la creación de un simulador para este tipo de productos, ya que el uso de estos aparatos tiene que ser seguro y estable, porque su pérdida puede originar graves consecuencias. Por lo tanto, la creación de un entorno de prueba para estos aparatos es necesaria.

Testear o simular un producto, es a día de hoy, algo indispensable en el mundo de la informática, ya que permite ver si el producto está listo para usarse en el mundo real. Durante el desarrollo de una plataforma de experimentación en un proyecto europeo, un fallo no experimentado en simulación ordinaria(SITL) junto con un corte de señal de control forzó una situación de riesgo que requirió una parada de emergencia, produciendo daños materiales en el equipo.

Por lo tanto, el departamento consideró necesaria la creación de un simulador mediante la técnica HIL o HITL(*Hardware-in-the-loop*)[1] en el entorno de simulación Gazebo[2] + ROS(*Robot Operating System)*[3].

HITL es una técnica de simulación que nos permite testear el firmware/software en el controlador de vuelo directamente, permitiéndonos trabajar directamente con él y con los actuadores y sensores del entorno simulado. Se usará junto a ROS para el envío de comandos y control de algoritmos y Gazebo como entorno de visualización 3D. Más adelante se entrará en detalle con todas estas tecnologías.

1.2 Actores implicados

En este proyecto hay una serie de personas implicadas, ya sea de forma directa o indirecta, desde el desarrollador hasta los beneficiarios, que disfrutaran de los desarrollos producidos en el proyecto.

1.2.1 Desarrollador

El desarrollador es la persona encargada de crear el simulador en el plazo de tiempo indicado. Normalmente, en un proyecto hay diversas personas que interpretan distintos roles(analista, programador, tester, diseñador, etc.), pero en este caso, el autor de este proyecto será el encargado de ejercer cada uno de estos roles.

1.2.2 Director y codirector

Tanto el director como el codirector son parte fundamental del proyecto, ya que deben guiar y supervisar el trabajo del desarrollador para que consiga cumplir los objetivos en el tiempo requerido. Además, también son beneficiarios directos, ya que podrán utilizar el trabajo para la obtención de nuevos proyectos.

1.2.3 Departamento

El departamento es el beneficiario principal de este proyecto, ya que sus integrantes y futuros estudiantes que hagan su trabajo de final de máster o grado utilizaran este simulador para testear sus trabajos en este ámbito y además dispondrán de una nueva herramienta que podrán usar tanto para formar nuevos alumnos como para continuar con el desarrollo de la investigación.

1.2.4 Empresas

Un beneficiario indirecto de este proyecto son las empresas implicadas en los proyectos que impliquen el uso de drones, ya que utilizarán el simulador para comprobar cualquier implementación y para garantizar la seguridad del proyecto.

Un ejemplo de este actor implicado es la EDAR de Sant Feliu de Llobregat. El departamento y la gestora de esta depuradora están trabajando en un proyecto que consiste en el uso de drones para recolectar agua de la planta, principalmente del decantador primario.

1.3 Estado del Arte

En este apartado se explicará estado actual de los drones y de sus simuladores.

1.3.1 UAV en la actualidad

Durante este último lustro, la palabra dron ha ganado una gran popularidad entre la población. Aproximadamente cada semana se puede leer una noticia de gran importancia en la cual este aparato está relacionado, ya sea por estar presente en uno de los proyectos más importantes de empresas muy influyentes(Amazon,Tesla,Google y Apple) o bien por "provocar" conflictos internacionales entre países[4].

Los UAV empezaron ganando popularidad durante la segunda década del siglo XX[5]. Su utilización era de carácter militar, y se usaban como blanco aéreo de entrenamiento. Con el inicio de la segunda guerra mundial, esta tecnología volvió a ganar importancia y se crearon drones con la capacidad de usar control remoto. Finalmente, durante la Guerra Fría, esta tecnología se desarrolló hasta el punto de poder realizar operaciones de reconocimiento.

Figura 1.1: Ryan Firebee, el modelo de dron soviético de los años 60.[6]

En los años 90, con la llegada del GPS y de la señal satélite, los UAV consiguieron volar más allá de la señal de control radio, gozando de mayor alcance y precisión a la hora de volar. Pero el gran boom en la sociedad se produjo a partir de 2010, ya que los drones empezaron a ganar gran popularidad entre la población, lo cual hizo que muchas empresas se fijaran en este producto y decidieron que esta herramienta era la solución a sus problemas o para optimizar su producto.

Actualmente, el nicho de mercado de los UAS es muy amplio[7]. Pueden tener una función logística, cargando y descargando mercancías entre diversos puntos, una función de reconocimiento, ya sea para carácter militar o de carácter agrícola, pudiendo monitorizar cientos de hectáreas mediante sensores. Además también se utiliza en el ámbito de la investigación, de la construcción o incluso del ocio.

Está claro que esta tecnología tiene un gran potencial y que es una herramienta que ha llegado para quedarse en el mercado. Sin embargo, todavía queda mucho trabajo por hacer, ya que su tasa de fiabilidad a dia de hoy no es igual a la de una aeronave tripulada y también trabajar el apartado ético, resolviendo preguntas sobre si limitar su venda, su actuación en las guerras y la utilización de inteligencia artificial en estos aparatos.

1.3.2 Técnica de simulación

Un simulador es un entorno simulado que nos permite controlar de distintas formas un vehículo modelado por un ordenador en un mundo virtual. Hay distintos tipos de simuladores, dependiendo la interacción de los distintos componentes y de si son componentes hardware o software. En este proyecto nos centraremos en 2 tipos: SITL y HITL.

El primer tipo de simulación, el SITL, es una técnica de simulación cuya particularidad es que no se necesita ningún tipo de hardware especial, es decir, no se necesita nada más que un ordenador, ya que simula cada uno de los componentes del circuito, ya sean sensores, actuadores, el firmware, etc.

En cambio, un simulador HITL es aquel en el cual se está testeando el firmware o el software en el sistema de forma que una pieza del sistema embebido es un componente hardware físico y está en comunicación con el sistema simulado.

Un ejemplo de simulador HITL sería el siguiente: Supongamos tener un sistema en el cual tenemos la controladora de vuelo, que tienes sus puertos de entrada salida. Esta controladora está ejecutando el firmware de control de vuelo y está conectada al sistema simulado mediante comunicación serial. Además, tiene conectado a un dispositivo de control remoto. Se usa este dispositivo indicando que el dron tiene que moverse hacia arriba. Esta información llega a la placa, que envía estos datos al simulador, que los recibe y utiliza los actuadores. Si todo va bien, veremos al drone moverse hacia arriba mientras que envía los datos de los sensores simuladores al controlador, creando un circuito cerrado.

Como se puede apreciar anteriormente, la principal ventaja del SITL es que no es necesario usar hardware y más simple de ejecutar, por lo tanto mejor para testear planes de vuelos iniciales. Mientras tanto, un simulador HITL requiere hardware,pero al usarse la placa se obtienen datos más reales en la simulación, y por lo tanto más similares a un vuelo real. Además. al usar el controlador y no tener que simularlo, se evita la sobrecarga de trabajo en equipo, permitiendo emplear toda la potencia en el entorno de simulación.

Por lo tanto, al tener una controladora de vuelo y un prototipo de simulador SITL ya creado en el departamento, se optó por dar enfoque a la técnica HITL, ya que de esa forma el departamento podrá testear sus modificaciones de forma más fidedigna y usarlo en la realidad con mayor seguridad. A continuación se muestra un esquema básico de cada tipo de simulador.

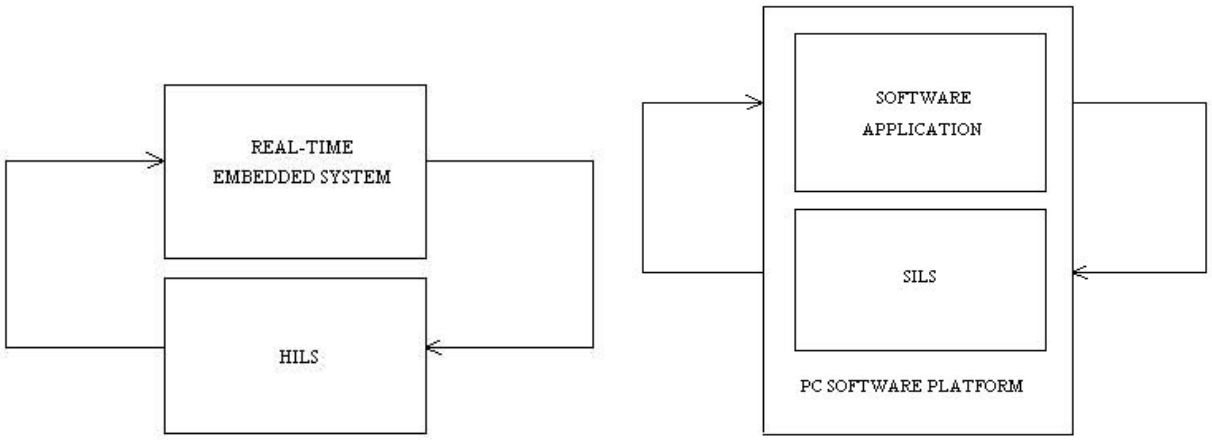

Figura 1.2: Comparativa entre HITL y SITL.

1.4. Proyectos similares

En esta sección se hablará sobre la frontera del conocimiento en el ámbito del proyecto. También se citan y se explican de forma breve los resultados de estudios anteriores.

1.4.1 Hardware in the Loop flexible para UAV

Este trabajo realizado en la UPC[8], concretamente en el departamento donde se realiza este trabajo, se decidió crear un modelo de simulador HITL para un UAV. Este modelo de simulador tiene la característica de ser modular e incorporar diversos nodos de ROS para ciertas tareas, como el mapeo y la navegación, ya que se desea que el dron trabaje en bosques densos y sea capaz de mapearlos y navegar por ellos.

Se hacen 5 propuestas: Una con un módulo hardware extra de Motion Capture, para mapear el terreno, otro usando el módulo ROS Octomap y otro usándose un módulo hardware SLAM(Simultaneous Localization and Mapping system). Las otras 2 propuestas son la unión de las 3 tecnologías y la unión del SLAM con el MoCap.

Este simulador se efectúa con Gazebo y ROS. Además, también aprovecha el plugin de Hector.

Cabe destacar que esto es un prototipo, pero aun así demuestra el potencial de los simuladores HIL/HITL.

1.4.2 SITL para Hybrid Drone

En este proyecto realizado en Corea del Sur[9], se desea crear un modelo de un UAV Híbrido, es decir, que tenga alas fijas y alas móviles. La principal característica de este proyecto es la creación de todos los algoritmos de control y movimiento de este modelo experimental y la creación e incorporación de un nuevo airframe para el simulador.

Además este proyecto utiliza las mismas herramientas que nuestro proyecto, salvando el tipo de simulación.

1.4.3 HIL para UAV con módulo de seguimiento de objetos

En este proyecto realizado por el departamento aeroespacial de Corea del Sur[10], se crea un simulador HITL en el entorno jMavSim. Se crea un simulador HITL capaz de crear objetos que junto a un algoritmo SURF(Speeded-Up Robust Features) y un algoritmo procesador de imágenes implementado en el dron, es capaz de captar los puntos interesantes del objeto y seguirle.

1.4.4 Conclusión

La conclusión de estos proyectos es que un simulador HITL es capaz de incorporar módulos ROS de diferentes categorías. Además, se puede apreciar que se puede testear de forma ideal un conjunto bien diferenciado de distintos algoritmos, ya sean algoritmos de vuelo o de control como algoritmos para detectar objetos y mapearlos mediante visión.

Por lo tanto se puede concluir que a día de hoy, se puede utilizar esta técnica en muchísimos ámbitos y que lo que hace diferenciar un simulador de estas características son los nodos/módulos externos que se utilicen y los algoritmos.

2. Alcance y Objetivo del Proyecto

En esta sección se hablará del alcance y de los objetivos y requerimientos del proyecto.

2.1 Objetivo

Como se ha mencionado anteriormente, el objetivo de este trabajo de final de grado es crear un simulador para drones. Para conseguirlo, se utilizará la técnica HITL y las tecnologías descritas en el apartado 2. La finalidad es conseguir un simulador adaptado a las necesidades del departamento y a la zona de trabajo del dron. También se valorará que sea fácil de utilizar. Además, también tiene que ser fácil de portar para que se pueda utilizar en el futuro.

2.2 Requerimientos

A continuación se establecen las necesidades, objetivos y condicionantes del proyecto.

- Las tecnologías que se utilicen en el proyecto han de ser de código abierto y algunas, como el uso de la Pixhawk, son impuestas por el departamento.
- El simulador tiene que ser fácil de portar.
- El simulador ha de ser fiable, es decir, dar datos correctos y que se puedan visualizar de forma sencilla, para así poder probar modificaciones y ver su funcionamiento.
- Generación de logs, es decir, ficheros que guarden un registro de todos los acontecimientos que se han producido en el sistema. Este fichero es muy útil para poder visualizar posibles problemas que sucedan en el modelo a testear.
- En todo momento se ha de establecer un feedback y un seguimiento con el director y codirector del proyecto.
- Acorde con el anterior punto nombrado, se necesitarán estos recursos humanos, es decir, se necesitarán aproximadamente unas 30 horas por parte del director y codirector y unas 490 horas por parte del estudiante.
- El trabajo debe estar realizado a finales de junio y comienza el 1 de febrero. Por lo tanto se tienen aproximadamente 5 meses para su realización.
- Se ha de poder utilizar cualquier tipo de sensor en este simulador y poder visualizar sus datos de forma sencilla.
- El simulador se ha de adaptar al entorno de trabajo del dron. Dado que los proyectos en este ámbito pueden ser múltiples, se ha de buscar una solución para poder crearlos y/o modificarlos de forma sencilla. Además, se creará un entorno similar a la EDAR de Sant Feliu de Llobregat ya que es el actual proyecto centrado en drones.

2.3 Riesgos

En esta apartado se mencionarán los riesgos que tiene el proyecto y sus posibles soluciones.

2.3.1 Problemas con el PC

Para ejecutar el simulador, se necesita un ordenador con altas prestaciones, que posea un buen procesador(mínimo una i5), una RAM de 8 GB como mínimo y una buena tarjeta gráfica.El acceso a este tipo de equipos es limitado dentro del grupo de investigación, por lo tanto, si el ordenador se rompe, esto supondría un problema.
Este problema tiene dos soluciones posibles: gestión de la garantía si sigue cubierto o adquirir uno nuevo. Dado que ambos casos suponen realizar múltiples gestiones tanto a nivel de universidad como otros actores, ambos supondrían un retraso significativo.

2.3.2 Problemas con el controlador

Al igual que con el PC, solo se posee un controlador de vuelo. Sin embargo, aquí hay 2 soluciones. O bien se intenta reorientar el TFG a la creación de un simulador SITL, que no necesita el controlador o bien se compra otro simulador, pasando por los mismos problemas descritos en el 3.3.1.

2.3.3 Aumento de tiempo no previsto por la realización de ciertas tareas

Hay actividades que tienen una cierta complicación, por lo tanto es posible que surjan complicaciones. Si surgiera este caso, el desarrollador tendría que dedicarle más horas de las que estaban previstas a estas tareas.

2.3.4 Cambio de requisitos

Puede ser que en mitad del proyecto, el departamento añada más requisitos o ideas para la creación del simulador. Esto crearía más trabajo que se tendría que adaptar a la planificación temporal.

En este caso, se estudiará la viabilidad del requisito. En caso de que sea viable, se dedicarán más horas diarias al proyecto para que no retrasen futuras tareas y conseguir que esta nueva tarea pueda encajar en la planificación temporal.

2.3.5 Desconocimiento de la tecnología

Como se puede apreciar en el apartado 2, el usuario tiene que enfrentarse a muchas tecnologías que no ha visto jamás. El avance del proyecto va ligado al grado de aprendizaje que se vaya adquiriendo a estas tecnologías. Por lo tanto el desarrollador tendrá que dedicarle horas al aprendizaje de las tecnologías que se usarán en el proyecto.

2.3.6 No disponibilidad del controlador

Al compartirse el controlador con otros estudiantes que están haciendo su proyecto, esto puede provocar que no se pueda usar porque otro estudiante ya está haciendo uso del mismo. Esto puede provocar retrasos en la planificación temporal. Para evitar este problema, se podría intentar pactar con los otros estudiantes el uso del controlador.

2.3.7 Errores en el firmware

En este proyecto se utiliza un firmware que es la encargada de controlar el UAV. Este firmware es de terceros y por lo tanto no está bajo nuestro control directo. Si hubiera algún fallo en este firmware que nos afectará, esto provocaría retrasos en nuestra planificación. Una posible solución para este caso sería adelantar otras funcionalidades mientras los desarrolladores del firmware arreglan el bug. Otra posible solución sería dar por no posible esta funcionalidad o bien buscar una solución alternativa.

2.4 Alcance

Antes de comenzar a implementar y construir el simulador, el estudiante tendrá que adaptarse y acostumbrarse con las distintas tecnologías/lenguajes que utilizará, que están detallados a continuación.

- **ROS**: El desarrollador tendrá que familiarizarse con ROS, especialmente con los nodos Mavros[11] y MAVLink[12].
- **Gazebo:** También se tendrá que adaptar al entorno de simulación Gazebo. Para ello, se intentara hacer los tutoriales de la web, para conocer y entender todas sus herramientas y sus posibles usos.
- **QGroundControl**: Se leerá la documentación de uso sobre este programa.

Una vez el desarrollador se haya adaptado y familiarizado con estas nuevas tecnologías, se dispondrá a usar el simulador SITL[13] utilizando como referencia el modelo de un estudiante. Todas los pequeñas modificaciones/experimentos tendrán que ser ejecutados en Gazebo y el estudiante verá su funcionamiento, haciendo pruebas y viendo su validez.

Después de este paso, y una vez que el estudiante se haya familiarizado con todas las tecnologías ya integradas, se procederá con la creación del simulador HITL. Primero se irán integrando poco a poco las tecnologías y una vez se hayan integrado todas estas tecnologías, se probará su integración en un pequeño test. Una vez esta fase este superada, se irá añadiendo modificaciones y se irá escalando el simulador, hasta que se logre todos los objetivos y requisitos del proyecto.

3. Metodología y Rigor

En esta sección se hablará de todo lo que incluye el alcance del proyecto, los objetivos principales, la solución propuesta, los métodos de trabajo aplicados para alcanzar la metodología elegida,las herramientas usadas y los riesgos que pueden aparecer y sus posibles soluciones.

3.1 Metodología

Para el desarrollo del proyecto se utilizará una metodología de desarrollo en cascada[14]. Este método de trabajo consiste en un proceso secuencial de diseño de software que tiene 5 fases, las cuales utilizaremos 4: Requisitos, Diseño, Implementación y Verificación.

No se incluye la fase de mantenimiento porque no nos alcanza en este proyecto y es el departamento quien se tiene que hacer cargo de ella.

En la fase de requisitos se analizan las necesidades del usuario final del producto y se llega a un consenso sobre todo lo que se requiere en el sistema y lo que no necesita o no requiere. Acto seguido, se pasa a la fase de diseño, donde se diseña como es el software. En esta fase además se deciden las tecnologías o los algoritmos que se van a utilizar en el programa.

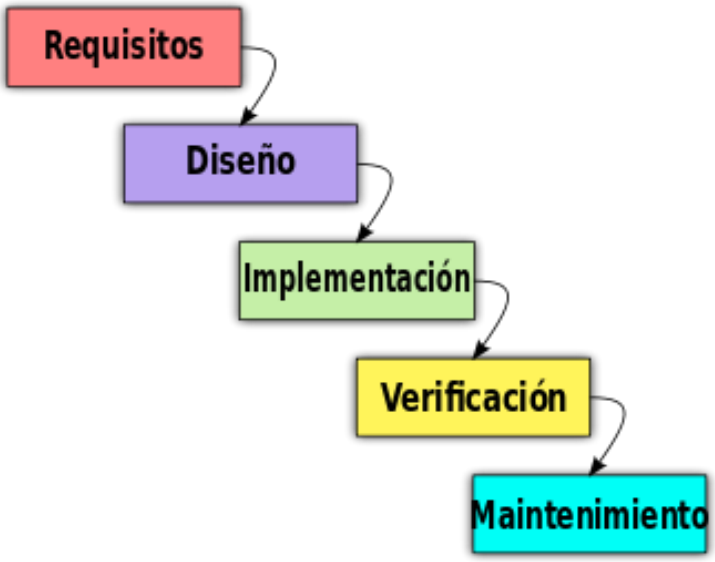

Figura 3.1: Desarrollo en Cascada

Cabe destacar que la fase de implementación y validez no será secuencial, sino iterativa, ya que se irán haciendo pequeñas y periódicas pruebas de validez sobre determinados objetivos. Por cada modificación del código y funcionalidad añadida a la implementación, se realizará un pequeño test

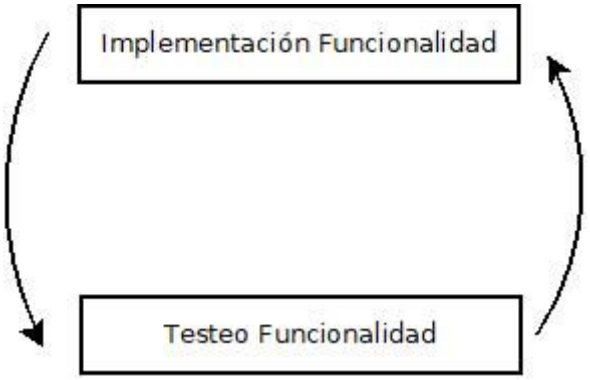

Figura 3.2: Fase de Implementación

La razón por la que se ha optado por una metodología de trabajo en cascada es porque al ser un trabajo de una persona, no se adecua a una forma de trabajo agile. A pesar de esto, sí utilizaremos un modelo de trabajo en paralelo en la implementación y validez, ya que al ser un trabajo muy costoso y largo, es conveniente hacer pequeñas pruebas para ir validando el trabajo poco a poco

Como parte importante de la metodología del trabajo, se intentará concertar reuniones quincenales con el director y/o codirector del proyecto, en las cuales se informará del estado del proyecto y de la validación de los objetivos del proyecto. A su vez, tanto director como codirector guiarán al desarrollador en el proyecto.

3.2 Herramientas de seguimiento

Al ser un trabajo de una sola persona, no hace falta pensar en herramientas que faciliten la compartición de código o ayuden al trabajo simultaneo. Para ayudar al control de versiones, se ha optado por usar Git[15], un gestor de repositorios que nos facilitaran la recuperación de datos y la disponibilidad de código.

Para el feedback con el departamento se intentará utilizar reuniones o charlas presenciales en el caso del codirector, ya que es el mejor método ya que se comportantes espacios en el laboratorio del grupo VIS. En el caso del director, la comunicación sera via correo electrónico.

3.3 Métodos de Validez

Se irá poniendo a prueba cada modificación que se haga mediante el uso de Gazebo + ROS. Gracias al entorno gráfico Gazebo, se puede apreciar si los comandos dados al controlador funcionan o no. También, gracias a Mavros y las herramientas de visualización ROS, podemos ver los datos de los sensores y ver si se produce una anomalía y en caso de existir, se utilizará el log generado para descubrirla.

Se intentarán crear juegos de prueba y casos para intentar descubrir el límite de recreación del simulador.

En cuanto a la validez de los requisitos y objetivos, tanto el desarrollador como codirector y director irán comprobando si estos mismos se van cumpliendo.

4. Planificación temporal

Este proyecto comprende desde el 1 de febrero hasta el 1-5 de julio, incluyendo memoria y defensa En los siguientes apartados, se detallarán las tareas que se tiene que realizar, en orden cronológico, con los recursos que se utilizarán y el tiempo estimado para cada tarea.

4.1 Tareas

En los siguientes apartados, se detallarán las tareas que se tiene que realizar, en orden cronológico, con los recursos que se utilizarán y el tiempo estimado para cada tarea.
El proyecto consta de unas 490 horas, que divididas por los días laborables (aproximadamente unos 110 días), se queda en unas 4,5 horas por dia.

4.1.1 Estudio y Análisis del proyecto

Antes de comenzar a implementar el simulador, el desarrollador debe analizar todas las tecnologías para decidir cuál es la más idónea para el proyecto. Para lograr este objetivo, el desarrollador estudiará y buscará información y documentación. Dado que muchas tecnologías las impone el departamento, esta tarea no requerirá más de 15 horas. Se utilizará un ordenador para esta tarea.

Una vez decididas las tecnologías que se utilizarán en el proyecto, el desarrollador y el director tendrán que pactar las tecnologías y crear un diseño inicial del proyecto. Para ello, se efectuará una reunión con el director para llegar a un consenso. Esta tarea no durará más de 5 horas y solo necesita recursos humanos.

Con los objetivos y los medios ya aclarados, como hemos dicho en el párrafo anterior, el estudiante tendrá que familiarizarse con el entorno. Para ello, el desarrollador realizará los tutoriales de todas las tecnologías desconocidas. Los recursos que se necesitarán son el ordenador del departamento y un total de 60 horas.

4.1.2 Gestión del proyecto

En esta tarea se incluye todo el trabajo referente a la gestión del proyecto, que tiene un trabajo total de 75 horas y se descompone de la siguiente manera:

- Alcance del proyecto y contextualización (26.0 horas)
- Planificación temporal (12.75 horas)
- Gestión económica y presupuesto (12.75 horas)
- Informe de sostenibilidad (12.75 horas)
- Reuniones y control del proyecto(10.75 horas)

La subtarea Reuniones y control de proyecto consta de 3 reuniones: una inicial, una intermedia y una final, que servirán para la corrección y planificación de la gestión del proyecto.

Los recursos que se utilizarán en esta tarea en todo momento serán Atenea, Gantter y Google Drive. En la subtarea reuniones y sesiones de control se hará junto al director y/o codirector.

4.1.3 Experimentación con el SITL

En esta fase del proyecto se creará el simulador SITL, se comprenderá su funcionamiento y se analizará el producto para ver sus errores y/o carencias y sus posibles mejoras. Adicionalmente, se comenzará con pequeñas implementaciones y optimizaciones, ya gran parte de estas mejoras serán portables al modelo HITL.

La razón por lo cual se construye un simulador SITL antes que un HITL es porque es más sencillo, ya que no requiere conexiones y está más documentado. Al ser una tarea menor, no se detallará en esta memoria.

Los recursos utilizados para esta tarea comprende desde todos los recursos software hasta la mayoría de los recursos materiales, teniendo en cuenta que no se utilizará la Pixhawk 2.8.4[16]. Se estima unas 85 horas de trabajo.

4.1.4 Simulador HITL

Finalmente llega la última parte técnica y larga del proyecto. Esta tarea comprenden desde la construcción del simulador hasta la parte de análisis del rendimiento.

Para empezar, se deberá construir el simulador HITL, para obtener un modelo básico. Esta tarea debería tardar unas 15 horas. Una vez logrado el modelo básico, comienza la parte de implementación del simulador.

Para empezar, se hará un diseño de lo que será el simulador. Para ello, se analizarán distintas fuentes de información y se volverá a efectuar una reunión con el departamento. Esta tarea solo necesita un ordenador y al director y/o codirector y al desarrollador y llevará unas 15 horas.

A continuación, se llevará a cabo la implementación y el testeo del simulador HITL, ya que como se dijo en la sección de metodología, se harán de forma paralela, implementando tests para cada funcionalidad implementada. Para esta actividad se necesitarán unas 120 horas, que se repartirán más o menos en 30 horas para pequeños test y comprobaciones y 90 para pura implementación. Para esta tarea se requerirá todo el material hardware y las tecnologías software.

Finalmente, se hará la implementación de unos tests exhaustivos y se probarán para la validación del estado del simulador. Estos tests serán una mezcla entre tests hechos por el desarrollador y tests proporcionados por Dronecode. Adicionalmente, también se incluirá en esta tarea la creación de un manual de instrucciones y de scripts para que el departamento pueda utilizar el simulador en un futuro. Por lo tanto, esta tarea necesitará unas 40 horas.

4.1.5 Memoria y Defensa

Como tarea final, se redactará la memoria final de este trabajo y se realizará una presentación para la defensa del proyecto delante del tribunal. Se estipula que tanto director como codirector ayudarán entorno a unas 10 horas al desarrollador en esta tarea.

Para esta tarea se utilizará Google Drive. El tiempo estimado para esta tarea es de 60 horas, que se repartirán en 45 para la redacción y 15 para la defensa ante el tribunal.

4.2 Recursos

En esta sección se detallarán los recursos humanos, software y hardware necesarios para la realización del proyecto. Destacar que todos los materiales software como hardware están disponibles en todo momento. Respecto a la disponibilidad de los recursos humanos, tanto director como codirector suelen estar en el lugar de trabajo del desarrollador, por lo tanto su disponibilidad es elevada.

4.2.1 Recursos humanos

- **Director y Codirector:** Ambas personas se encargarán de guiar y supervisar el trabajo del desarrollador para que se cumplan todos los objetivos.
- **Personal del Departamento:** Ayudarán al desarrollador y le aconsejarán.

- **Desarrollador:** El encargado de todo el proyecto. Tiene que conseguir que todos los objetivos del proyecto se cumplan.

4.2.2 Recursos software

- **Sistema operativo**: El proyecto se desarrollará en el sistema operativo Ubuntu 16.04. Sin embargo, también se usará un PC con Windows 10 Home, para la documentación y escritura de las entregas de GEP.
- **Control de las versiones**: Se usará Git.
- **Editores**: Se utilizará Google Drive[17] para la escritura de las entregas y de la memoria. Para la edición de código, se utilizará vim.
- **Gantter y Trello**: Para la planificación temporal se usará Gantter[18], mientras que se usará Trello[19] para gestión del trabajo.
- **Gazebo**: Se empleará Gazebo como entorno de simulación para el proyecto
- **PX4**: El autopilot PX4[20] se utilizará para la simulación del controlador aéreo.
- **ROS**: Se empleará este framework para la comunicación con el controlador.
- **QGroundControl**: Se maneja este software como estación de control de drones.
- **Blender**: Se utilizara Blender para manejar archivos Collada de forma que se pueda utilizar en Gazebo.

4.2.3 Recursos materiales

- **Lugar de trabajo**: Para hacer el proyecto de la forma más adecuada y cómoda posible, el departamento facilita una de las mesas de la sala. Además, también se tiene que tener en cuenta otros recursos indirectos como luz, calefacción, cafetera, microondas y muchos más.
- **Ordenador personal**: Un Lenovo IdeaPad 320 junto a un ratón, para la escritura de la documentación y de la memoria.
- **Ordenador de Trabajo**: Un ordenador de altas prestaciones. Es un HP Omen con 16 GB de RAM, una tarjeta gráfica NVIDIA GTX 1050 y un procesador Intel Core i7-8750H.
- **Pantalla**: El departamento proporciona una pantalla para poder trabajar de una forma más eficiente, debido a la gran cantidad de programas que tendrán que estar abierto simultáneamente. El producto en cuestión es una pantalla Benq BL2045PT.
- **Pixhawk 2.8.4**: El controlador aéreo del dron, necesario para poder utilizar la técnica HIL para el simulador.
- **Futaba T8J**: Emisora de 8 canales para aviones, veleros y helicópteros[21]. Viene con el receptor Futaba R2008SB.

4.3 Diagrama de Gantt

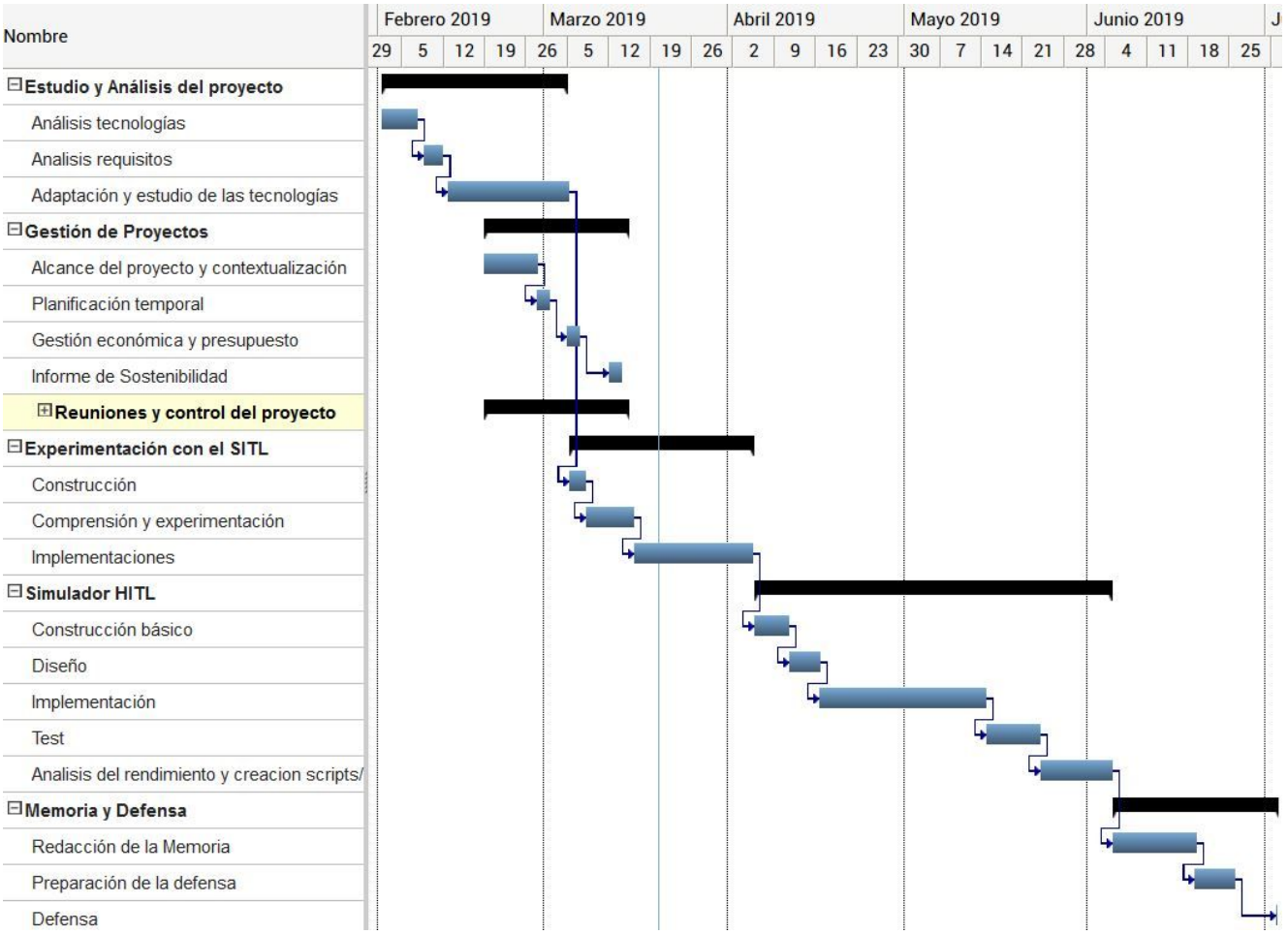

Figura 4.1: Diagrama de Gantt

4.4 Predecesores y Duración

Nombre	Duración	Inicio	Fin	Predecesoras
Estudio y Análisis del proyecto	**21.88días?**	**02/01/2019**	**03/04/2019**	
Análisis tecnologías	4.5días?	02/01/2019	02/07/2019	
Analisis requisitos	1.63días?	02/08/2019	02/11/2019	2
Adaptación y estudio de las tecnologías	14.88días?	02/12/2019	03/04/2019	3
Gestión de Proyectos	**19días?**	**02/18/2019**	**03/14/2019**	
Alcance del proyecto y contextualización	7.5días?	02/18/2019	02/27/2019	11
Planificación temporal	3días?	02/27/2019	03/01/2019	6
Gestión económica y presupuesto	3días?	03/04/2019	03/06/2019	7
Informe de Sostenibilidad	3días?	03/11/2019	03/13/2019	8
Reuniones y control del proyecto	**19días?**	**02/18/2019**	**03/14/2019**	
Reunion Inicial	1día?	02/18/2019	02/18/2019	
Reunion Intermedia	1día?	03/04/2019	03/04/2019	7
Reunion Final	1día?	03/14/2019	03/14/2019	9
Experimentación con el SITL	**23días?**	**03/04/2019**	**04/04/2019**	
Construcción	2.63días?	03/04/2019	03/07/2019	4
Comprensión y experimentación	6.38días?	03/07/2019	03/15/2019	15
Implementaciones	14días?	03/15/2019	04/04/2019	16
Simulador HITL	**42días?**	**04/04/2019**	**06/03/2019**	
Construcción básico	3.5días?	04/04/2019	04/10/2019	17
Diseño	3.5días?	04/10/2019	04/15/2019	19
Implementación	20días?	04/15/2019	05/13/2019	20
Test	7días?	05/13/2019	05/22/2019	21
Analisis del rendimiento y creacion scripts/manual	8días?	05/22/2019	06/03/2019	22
Memoria y Defensa	**20.12días?**	**06/03/2019**	**07/01/2019**	
Redacción de la Memoria	10días?	06/03/2019	06/17/2019	23
Preparación de la defensa	4.5días?	06/17/2019	06/24/2019	25
Defensa	1día?	07/01/2019	07/01/2019	26

Figura 4.2: Tabla de predecesores y duración de las tareas

4.5 Plan de actuación

Probablemente durante el proyecto surjan desviaciones respecto a la planificación, ya sea porque se acaba una tarea antes de tiempo o si la tarea dura más de lo previsto.
Si sucede este primer caso, no pasa absolutamente nada malo, simplemente se adelanta la tarea siguiente, quedando esas horas libres por si surgen horas extras. En caso de que el adelanto sea superior a los 4 dias, se reestructurará la planificación, seguramente añadiendo unas 8 horas a la etapa con más riesgo de retrasarse, qué es la implementación del HITL, y adelantando 2 dias la planificación.

Para evitar el segundo caso, se ha optado por añadir horas de trabajo extras a cada tarea que en principio no son necesarias y que se pueden utilizar en caso de que la tarea se alargue. Adicionalmente, también se dispone una semana extra(5 días) que se pueden usar en caso de retraso. Por lo tanto, se prevé que si no hay una gran cantidad de desviaciones, se pueda entregar el producto antes del tiempo previsto.

Si el desvío de la planificación es demasiado elevado, y en alguna tarea se supera ese margen de 20 horas, se tendrá que reestructurar la planificación temporal. Si se supera el margen en una tarea que no sea Gestión de Proyectos, Simulador HITL o Memoria y Defensa, se procederá a la siguiente tarea, ya que no se consideran tareas críticas. En el caso de las tareas anteriormente mencionadas, al ser críticas,o bien se aumentará el número de horas dedicadas a la tarea, aproximadamente unas 20, o bien se tendrá que hacer una simplificación de la misma, siempre en consenso con el director, con el cual se tendrá que hacer una reunión de 2 horas.

Gracias a estas medidas, plan de acción y planificación temporal, el desarrollador se asegura de que el proyecto terminará en el plazo establecido.

4.6 Desviaciones

Durante el transcurso de este proyecto no se han producido grandes variaciones temporales. Sin embargo, a la hora de hacer la construcción básica y el diseño del simulador HITL, hubo tres problemas que provocaron desviaciones en la planificación temporal.

4.6.1 Error en la placa

Nuestra placa Pixhawk 2.8.4 sufría un error que no permitia ejecutar una simulación HITL. Esto es debido a que las primeras placas de este diseño tenian un error de hardware que provocaba que solo se aprovechara un MB flash en lugar de los 2MB que poseía. Además, el firmware destinado para estas placas, el PX4_FMU V2, solo aprovechaba un MB de memoria flash. Arreglaron este error lanzando una versión V3 del firmware y actualizando el bootloader del QGC para reconocer las placas[22].

Se estudiaron 2 soluciones: Instalar la v3 si nuestra placa no tenía el microprocesador causante del error, el STM32F427VIT6, o bien si esta hipótesis era errónea, cargar el módulo *pwm_sim_out,* el encargado del HITL, que estaba desactivado por este problema de capacidad.

Después de comprobar que nuestra placa tenía otro microprocesador, el departamento y el desarrollador se decantaron por la primera opción. Se actualizó el bootloader para que descargara la versión v3 en lugar de la v2 y de paso de paso de la versión 1.7.2 a la versión 1.9.0.

En este proceso se perdieron unas 5 horas.

4.6.2 Puerto UDP no operativo

Durante la simulación SITL, se utiliza el puerto UDP 14540 para comunicar el nodo MAVROS con el autopilot. Sin embargo, por razones desconocidas, esto no funciona con el HITL. Como solución, se intentó usar el paquete mavlink-router[23] de Intel para enviar paquetes UDP de 15440 al puerto indicado mediante el comando. Sin embargo, aunque los paquetes llegarán, el autopilot los rechazaba.

Como solución a este problema, se decidió utilizar el puerto del QGC, el 15450 para utilizar MAVROS. Esto fue exitoso, aunque esto provocará que no se puedan utilizar simultáneamente QGC y MAVROS.

Finalmente, y tras advertir a los empleados de PX4 del error, éste se solucionó en la versión 1.9.0 del firmware.

En este proceso se perdieron unas 15 horas.

5. Gestión Económica y Sostenibilidad

En esta sección se efectúa una identificación del coste de los recursos y con ello, un presupuesto. Además también se explica cómo se fraguara el control de la gestión económica. FInalmente, se argumenta la sostenibilidad de este proyecto desde tres sectores diferentes: social, económico y ambiental.

5.1 Identificación y estimación de los costes

En este apartado se realiza una identificación y estimación de los costes de todos los elementos que componen el proyecto. Se identificarán los costes de los recursos humanos, recursos software, recursos hardware, costes indirectos, imprevistos y contingencia.

5.1.1 Costes Humanos

Este proyecto lo ha realizado una persona, que ha adoptado diferentes roles en el proyecto. A continuación, se muestra una tabla con los precios por hora de cada perfil, basado en el salario bruto[24]. Consideramos que la tarea Gestion de Proyecto la efectúa el rol de analista y que director y codirector efectuarán unas 30 horas en reuniones.

Rol	**€/h**	**Estimación horas**	**Total**
Director	50	30	1.500€
Codirector	50	30	1.500€
Analista	35	120	4.200 €
Programador	25	310	7.750 €
Tester	25	70	1.750 €
Total		**560**	**16.700 €**

Tabla 5.1: Costes Humanos

5.1.2 Costes directos por Actividad

En la siguiente tabla, la tabla 5.2, se especifica el coste directo de cada actividad, siguiendo el diagrama de GANTT(Figura 3)

Nombre Actividad	Horas Estimadas	Recurso	Coste Total
Analisis Tecnologías	15	Analista	525€
Análisis Requisitos	5	Analista Director CoDirector	175€ 250€ 250€
Adaptación y Estudio Tecnologías	60	Programador	1.500€
Alcance Proyecto y Contextualización	26	Analista	910€
Planificación Temporal	12.75	Analista	446,25€
Gestión Económica	12.75	Analista	446,25€
Informe Sostenibilidad	12.75	Analista	446,25€
Reunión y Control Proyecto	10.75	Analista Director CoDirector	376,25€ 537,5€ 537,5€
Construcción SITL	5	Programador	75€
Compresión y Experimentación	20	Programador	500€
Implementación	60	Programador	1.500€
Construcción HITL básico	15	Programador	375€
Diseño	15	Analista Director CoDirector	525€ 750€ 750€
Implementación	90	Programador	2.250€
Test	30	Tester	750€
Análisis rendimiento y creación scripts y manual	40	Tester	1.000€
Redacción memória	45	Programador	1.125€

		Director CoDirector	500€ 500€
Defensa	15	Programador	375€
Total			**16.700 €**

Tabla 5.2: Costes directo por Actividad

5.1.3 Recursos Hardware

En este apartado se presentan los distintos elementos hardware utilizados durante el proyecto. Se les estima una vida útil de 4 años. Para el cálculo de la amortización, se decidió calcular de la siguiente forma, contando un trabajo diario de 8 horas y 260 días laborables al año.

Amortización(€/h) = Precio / 4 años * 8 horas * 260 días
Amortización(5 meses) = Amortización(€/h) * (260*5/12) días * 8 horas

Recurso Hardware	**Precio(€)**	**Vida Útil**	**Amortización(5 meses)**
HP Omen	1300 €	4 años	135,46 €
Lenovo Ideapad 320	650 €	4 años	67,73 €
Pantalla Benq	100 €	4 años	10,42 €
Pixhawk 2.8.4	90 €	4 años	9,4 €
Mando Dualshock 3	50	4 años	9.21 €
Total			**232,21 €**

Tabla 5.3: Costes de los Recursos Hardware

5.1.4 Recursos Software

En este apartado se presentan los distintos elementos software utilizados durante el proyecto. La mayoría de ellas son gratuitas. La amortización se calcula de la misma forma que en la de recursos hardware. Se estima una vida útil de 5 años.

Recurso Software	**Precio(€)**	**Vida Útil**	**Amortización(5 meses)**
Windows 10 Home	145 €	5 años	13 €
ROS	Gratuito	-	-
QGroundControl	Gratuito	-	-
Gazebo	Gratuito	-	-
Gantter	Gratuito	-	-
Ubuntu 16.04	Gratuito	-	-
PX4	Gratuito	-	-
Editores(Vim/Drive)	Gratuito	-	-
Git y Trello	Gratuito	-	-
Blender	Gratuito	-	-
Total			**13 €**

Tabla 5.4: Costes de los Recursos Software

5.1.5 Costes Indirectos

En este apartado se tratan todos los costes o facturas que están asociados al proyecto de forma indirecta, tales como gasto energético o el precio de la oficina como el precio del transporte. El precio de la oficina/sala no se ha podido cuantificar, por eso se ha hecho una estimación. Para la luz se ha estimado un gasto de 0.7 kwh a la hora.

Coste Indirecto	**Coste(€)**	**Periodo/Unidades**	**Coste Final**
Transporte(T-Jove)	105 €	2 unidades	210 €
Acceso Internet	30 €	5 meses	150€
Luz	0.12 kwh/€	490 horas	41 €
Gastos Oficina	60 €	5 meses	240 €
Total			**641 €**

Tabla 5.5: Costes Indirectos

5.1.6 Contingencia e imprevistos

Los 2 mayores imprevistos y riesgos que pueden suceder en este proyecto son el retraso en la planificación, que puede implicar horas extra y en el surgimiento de algún fallo en el pc Omen, necesario para el trabajo o en la Pixhawk 2.8.4. Si hubiera algún fallo en los otros elementos hardware, no haría falta un gasto extra de presupuesto, ya que no son imprescindibles e incluso garantía, como es el caso de la pantalla.

Para el primer riesgo, el de la planificación, se estipula en un 15%. Este riesgo puede implicar, como ya se mencionó en el apartado Desviación y Plan de Actuación, unas 20 horas de trabajo extra para el desarrollador y unas 2 horas para el director. Por lo tanto, el cálculo del imprevisto es de (25€ /h* 20 h + 50€ /h * 2 h) * 0.15, que da un resultado de 90 euros.

El segundo riesgo se estima sobre un 10% en la Pixhawk como el HP Omen. Por lo tanto el cálculo es su precio por la probabilidad por cada producto, lo que da un resultado de 140 euros. En el caso del mando PlayStation, al ser un producto que se utilizara con menos frecuencia se estima un 5% de riesgo, añadiendo unos 12 euros a esta partida del presupuesto, quedando un total de 152 euros.

Por lo tanto, el cálculo total de esta partida se queda en unos 242 euros.

En cuanto a la contingencia, al ser un presupuesto detallado y preciso, se considera que un 5% es un margen óptimo. Otro argumento a favor de este porcentaje es que el riesgo percibido es bajo, ya que los recursos hardware más caros ya tienen un costo de imprevisto y en la planificación hay muchos mecanismos para evitar una desviación.

5.1.7 Coste total

Concepto	**Coste**
Costes directos	16.945 €
Costes indirectos	641 €
Coste Imprevistos	242 €
Contingencia(5%)	891 €
Total(IVA incluido)	**18719 €**

Tabla 5.6: Coste total

5.2 Control de la gestión

Una de las mayores desviaciones en los proyectos de carácter informático suele ser la desviación de horas en la duración de las distintas actividades, especialmente las de implementación. Para evitar esta problemática, se realizará un seguimiento diario en las horas invertidas de las correspondientes tareas y se irán sumando y comparando con las horas estimadas para poder detectar desviaciones, especialmente en la parte de implementación del HITL, ya que es la parte con más riesgo. Al finalizar el proyecto, se utilizara la siguiente fórmula para calcular la desviación definitiva en horas del proyecto:

-Desviación en el coste de horas: (horas estimadas –horas reales) * coste por hora.

Otra ventaja de esta estrategia es que gracias a la recopilación de estos datos, se podrá hacer mejores estimaciones de tiempo en futuros proyectos de características similares a este.

Además, se utilizará Trello para llevar una lista TO-DO. Con esta herramienta se marcarán las tareas pendientes con su fecha de finalización y también habrá una lista con las tareas acabadas y su día de finalización. Finalmente, también se harán reuniones de seguimiento quincenalmente con el director o subdirector para evitar desviaciones.

Adicionalmente, también se incluye una partida en el presupuesto en forma contingencia e imprevistos, que nos aseguran tener un colchón por si surge alguna desviación.

5.3 Desviaciones del Presupuesto

Como se ha mencionado en el apartado Desviaciones, este proyecto ha tenido un incremento de 20 horas respecto a la planificación temporal. Estas horas se tienen que pagar, a precio de programador. Por lo tanto la cantidad que nos sale es 25*20, cuyo resultado es 500 euros.

Además. se optó por utilizar el Mando Futaba T8J, cuyo precio es de 250 euros y su coste no se preveía en el presupuesto. El cálculo de la amortización es de 30 euros, considerando que se utilizará 5 meses y que su vida útil es de 4 años.

Por lo tanto, el presupuesto contó con 530 euros extra, que sin embargo, están cubiertos por la contingencia, ya que su valor es de 891 euros.

6.Sostenibilidad y compromiso social

En este apartado se efectúa un análisis de la sostenibilidad de este proyecto. Para eso, se ha seguido el modelo de la matriz de sostenibilidad proporcionada por la Facultad de Informática de Barcelona.

6.1 Dimensión económica

En este proyecto se ha realizado una evaluación de todos los costes, tanto materiales como humanos, junto a los posibles imprevistos, como se puede ver en el apartado Identificación y estimación de costes.

En cuanto a recursos humanos, si es posible que este proyecto tuviera un coste menor, ya que el desarrollador tiene que familiarizarse con las tecnologías. Este tiempo remunerado no sería necesario en caso de haber contratado a un programador ya experimentado.

En cambio, es inviable realizar el trabajo con menos tecnologías. En el tema hardware, se podría prescindir de la pantalla extra. Sin embargo, este elemento aumenta la eficiencia del trabajo, por lo tanto si se quitara, aumentaría el número de horas en cada tarea, quedando un presupuesto similar. Por lo tanto, el presupuesto es ajustado, viable y competitivo.

Fuera del ámbito presupuestario, cabe destacar que este proyecto tiene una influencia positiva en el departamento, y por extensión, en la UPC. La herramienta construida en este trabajo reduce el porcentaje de riesgo de accidentes de los drones, y por lo tanto, hace que los gastos por reparación y horas extras del personal tengan que ser menos. Por ejemplo, el accidente explicado en el apartado Contexto se hubiera podido evitar en el caso de haber tenido este simulador, ya que se hubiera comprobado todo de una forma más segura, y así haberse evitado estos costes adicionales.

Además, al poder simular un entorno y los sensores, este proyecto puede ayudar a estimar presupuestos en proyectos en fases tempranas, ya que ayuda a ver lo que es necesario y lo que no es necesario.

Existe un riesgo en este proyecto y es que PX4 deje de dar soporte al simulador HITL. En este caso nos tendríamos que quedar para siempre con la última versión compatible. En tema actualizaciones, es necesario que cada 5 años se cambie el PC y la placa, ya que habrán agotado su vida útil.

6.2 Dimensión social

A nivel personal, este proyecto supone una oportunidad ideal, ya que me sumerge en un nuevo ámbito, el mundo de la robótica. Como se puede apreciar en la planificación temporal, el desarrollador verá y aprenderá bastantes nuevas tecnologías interesantes. Además de ver el mundo de la robótica, también tendré la oportunidad de ver como funciona un firmware de controlador de vuelo, aprender Gazebo y adentrarme en el mundo de los UAS.

Este proyecto permitirá a la gente del departamento testear sus implementaciones de forma ideal y dará mucha más seguridad a la hora de probarla en el dron real, por lo tanto beneficiará específicamente al departamento y a la UPC. Además, esta herramienta abre nuevas oportunidades educativas para que los alumnos trabajen con herramientas reales sin riesgos y costes muy bajos.

Finalmente, añadir que todo este proyecto se ha utilizado con tecnologías de código abierto y gratuitas. El proyecto se subirá a Github de forma pública, dando la oportunidad para que la gente que quiera ver este sector vea este proyecto y aprenda con él. Esto beneficiara a este sector, sin embargo podría perjudicar al sector de freelancers o profesores que lanzan tutoriales similares, aunque se calcula que el riesgo de esto sería ínfimo.

6.3 Dimensión ambiental

La elaboración del proyecto provoca la utilización de diferentes recursos hardware que consumen electricidad/luz. El coste de este tipo de energía tiene un impacto ambiental, que depende de su método de obtención. Para este proyecto, se tiene que mantener alimentado un portátil, una pantalla extra y un controlador de vuelo, por lo tanto el gasto medio se estima sobre unos 0.7 kwh.

La gran mayoría de los recursos utilizados en el proyecto están siendo o pueden ser reutilizados. El controlador ya se ha utilizado en anteriores proyectos y se utilizará en futuros, al igual que el mando Futaba y el portátil HP. También recalcar que este proyecto podría evita accidentes, ergo podría evitar el consumo de nuevos recursos dedicados a la reparación.

La vida útil del producto desarrollado es bastante elevada, ya que es muy portable a futuras versiones del firmware PX4 y por lo tanto, portable a otros controladores de vuelo que el departamento decida comprar.

Sin embargo, cabe añadir que el coste energético de este proyecto depende en parte de la cantidad de sensores que no se están simulando y del consumo energético del controlador, por lo tanto si se comprara una placa más potente y que consuma más, el simulador consume más energía. En el lado positivo, destacar que este proyecto será muy reutilizado, ya que en cada proyecto que implique el uso de un drone se usará el simulador.

Por último, recalcar que un simulador SITL consume menos que un HITL, debido a que no se usa la controladora ni otro tipo de sensores. Sin embargo, antes se pasa de SITL a uso real, y ahora al tener un simulador HITL en el cual se pueden testear sensores físicos, se evita el uso real, evitando accidentes y el uso de la batería, el componente más tóxico del sistema.

7.Tecnologías Utilizadas

En este apartado se describe todas las tecnologías utilizadas en este proyecto. Se detalla también el porqué del uso de estas tecnologías y no de otras, ofreciéndose comparaciones en ciertos casos.

7.1 Ros

ROS es un middleware, en otras palabras, consiste en un conjunto de herramientas, nodos, librerías y APIs, empezando por drivers y acabando por algoritmos avanzados, que componen un framework de programación orientado a la robótica. Su objetivo principal es la reutilización de código en investigación y desarrollo robótico, y además es software libre bajo la licencia BSD, la cual permite libertad para uso comercial e investigador. En este proyecto se usará ROS Kinetic.

ROS además posee servicios como abstracción del nivel hardware, mantenimiento de paquetes, paso de mensaje entre paquetes y control de dispositivos de bajo nivel.

ROS tiene tres niveles conceptuales: el nivel de sistema de ficheros, nivel computacional de grafos y nivel de comunidad.

7.1.1 Nivel Sistema de Ficheros

A continuación se define la estructura y los componentes del sistema de ficheros de ROS:

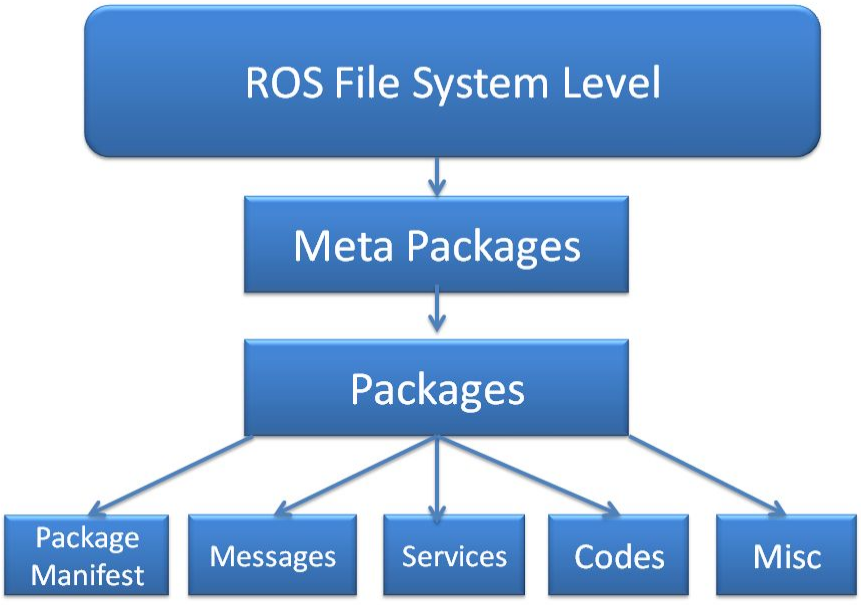

Figura 7.1: Niveles del sistema de ficheros de ROS.

- **Paquete:** El paquete es la unidad mínima y granular de organización de ROS que se puede construir.Los paquetes están compuestos de uno o más procesos ejecutables(nodos), por la librería de dependencias, ficheros de configuración como los manifiestos, datasets y demás archivos útiles para la organización.
- **Metapaquetes**: Los metapaquetes son paquetes especializados que unen diversos paquetes.
- **Manifiesto del Paquete**: Los manifiestos son archivos .xml que alberga información muy variada del paquete. Esta información va desde el nombre del paquete, su creador, la versión actual y descripción, hasta las dependencias, ya sean de sistema o de paquetes. También incluye el tipo de mensajes que tiene este paquete y sus licencias.
- **Mensaje y Servicios**: Descripción de los mensajes y los servicios.

Los paquetes se construyen mediante catkin, un sistema de construcción que utiliza python y CMake. Por lo tanto, el paquete tendrá un fichero CMake que contiene datos como la versión mínima de cMake, nombre del paquete, módulo python, adicción de paquetes y librerias, servicios y mensajes que se utilizan y tests a ejecutar.

7.1.2 Nivel computacional de grafos

Este nivel de ROS es en realidad una red *peer-to-peer,* lo cual quiere decir que es una red en la cual todos los nodos que la componen se comportan al mismo nivel y por lo tanto, pueden recibir como dar información entre ellos.

Los principales elementos de este nivel son:

- **Nodos**: El principal elemento de computación de ROS. Este elemento normalmente se encarga de controlar alguna funcionalidad robótica.
- **Nodo Master:** Este nodo es el más importante de todo, ya que es el encargado de la comunicación entre nodos. Su principal función es la de proveer registro a los otros nodos, es decir, permite que los nodos publiquen o escuchen topics.
- **Mensajes**: El elemento de comunicación. Un mensaje es simplemente una estructura de datos, compuesta por tipo de datos primarios, como puede ser floats, integers o chars, como arrays de los mismo e incluso structs.
- **Topics:** Los topics son los buses por los cuales los nodos intercambian mensajes. Estos buses reciben un nombre, que es el que usará el nodo Master para conectar a los nodos publicador y suscriptor a este topic. Cabe destacar que múltiples nodos pueden recibir o enviar mensajes a un topic, y que la comunicación es asíncrona.

- **Servicios**: Los topics son útiles para comunicaciones entre muchos, sin embargo no lo son para una comunicación de solicitud y respuesta. Hay 2 tipos de mensajes: uno para solicitud y otro para respuesta. Un nodo envía la solicitud de respuesta y espera hasta recibir la respuesta, lo cual establece una comunicación síncrona.
- **Bags**: Los bags son ficheros que guardan todos los datos de ROS se han usado en una ejecución. Estos datos son importantes a la hora de recrear, y analizar ejecuciones anteriores.

7.1.3 Paquetes ROS

Durante el transcurso de este proyecto, se han utilizado diversos paquetes de ROS, los cuales se describen a continuación.

- **gazebo_ros_pkgs:** Este metapaquete está compuesto por diversos paquetes que contienen los adaptadores necesarios para simular un robot en Gazebo utilizando ROS.

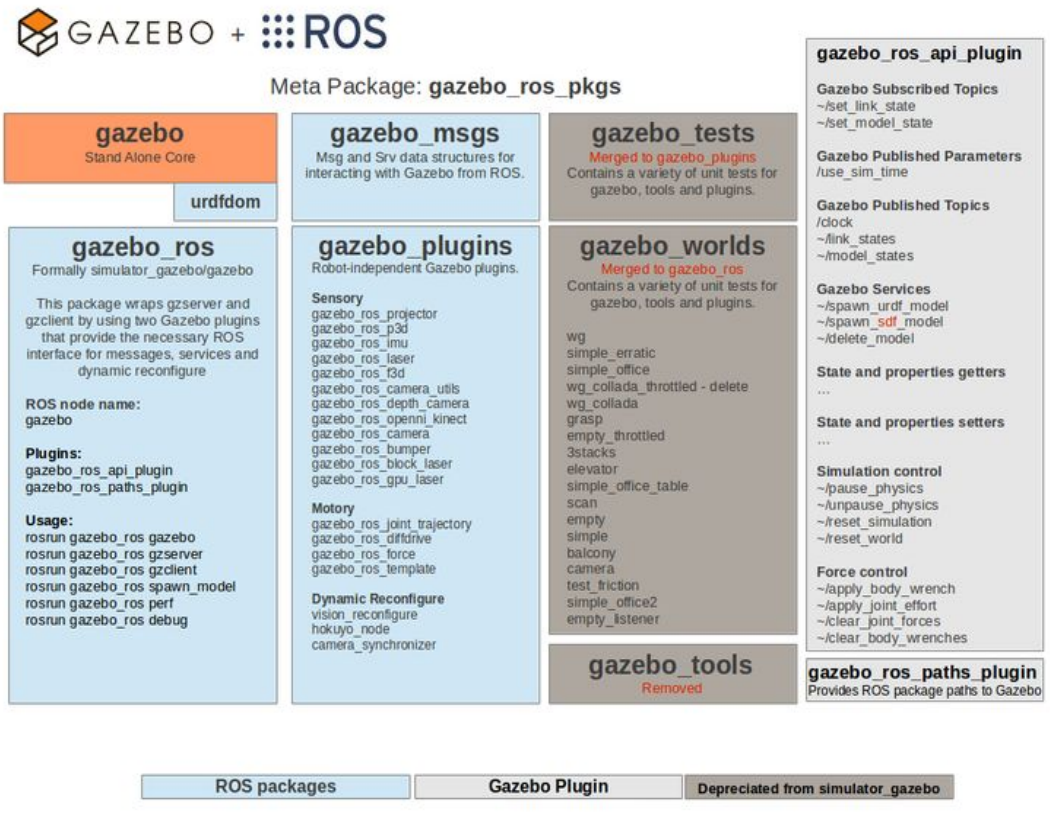

Figura 7.2: Paquetes y plugins de gazebo_ros_pkgs[25].

- **tf**: Paquete encargado de realizar las transformaciones de las distintas partes que componen nuestro dron respecto a un punto de referencia[26].
- **robot_state_publisher**: Es el nodo responsable de publicar la posición del dron en el TF en cualquier instante.
- **joint_state_publisher**: Este nodo publica el estado y ángulo de las uniones del dron en el robot_state_publisher.
- **Mavros**: Uno de los nodos más importantes de este proyecto. Este nodo permite usar el protocolo de comunicación estándar para UAV Mavlink en ROS, provocando que mediante ROS podamos enviar comandas al dron y recibir datos del mismo.

7.1.4 Herramientas ROS

- **Rviz:** Herramienta de visualización de ROS. Nos permite visualizar ciertos sensores del drone como cámara, láser y lidar. También nos permite visualizar el camino recorrido del dron, su modelo y muchas más cosas[27].
- **rqt**: Metapaquete de ROS compuesto por múltiples herramientas que nos permiten la visualización de ciertos elementos. Nos permite desde ver la visualización de cámaras(rqt_image_view), gráficos de ciertos topics numéricos(rqt_plot) pasando por generar y manejar rosbags (rqt_bag) hasta permitir ver grafos de la arquitectura de nodos y topics que se están ejecutando en ese momento(rqt_graph y rqt_tf_tree).
- **rosnode**: Esta herramienta nos permite ver los topics a los cuales está suscrito y está publicando y sus mensajes.

ROS tiene su propia línea de comandos que nos permiten movernos con rapidez entre los distintos paquetes y nos permite ejecutar fácilmente ROS. Los comandos son los siguientes:

- **rosls y roscd**: Nos permiten la visualización y el movimiento entre los directorios ROS.
- **rosrun [Nodo]**: Nos permite ejecutar el nodo ros que queramos.
- **roscore**: Ejecuta el nodo maestro de ROS.
- **rostopic + opción**: Si incluimos la opción echo, nos permite visualizar los mensajes que se están transmitiendo en ese topic. Con la opción list, visualizamos la lista de topics.
- **roslaunch:** Ejecuta ficheros del tipo .launch.

7.2 QGroundControl

Una estación de control de tierra para drones es un software que monitorea y supervisa el UAS. Cuenta con un sistema de observación para realizar el análisis (generalmente gráfico) de la información adquirida, además de mecanismos para el control del UAV.

En nuestro caso se utilizara el software QGroundControl. Es un programa que permite el setup del stack PX4, y que permite enviar órdenes de vuelo, calibrar los sensores e incluso dirigir al dron mediante un joystick. Se decidió utilizar esta tecnología porque es la única que se podía utilizar con el pilot PX4, con MAVLink y que funciona de forma gráfica, y por lo tanto, más sencilla de usar. En la tabla 7.1 se ofrece una comparativa donde se puede apreciar las características de cada software de este estilo examinado.

Este programa nos ofrece las siguientes posibilidades:

- Posibilidad de visualizar cada Mavros topic en tiempo real mediante el uso del widget Analyze en forma de gráficos, como se puede apreciar en la figura 6.4.
- Cambio de los parámetros de vuelo del UAV, algunos incluso con el UAV en vuelo. Se puede modificar configuraciones como el estimador de posición, decidir qué sensores usar de forma real (no simulados), setup de seguridad y emergencia, etc.
- Visualización en 2D del dron. También ofrece la posibilidad de configurar waypoints en el mapa mediante arrastrar y soltar, como se puede ver en la figura 6.3.
- Multiplataforma(Windows, Linux, Mac, Android e IOS).

GCS Software	**QGroundControl**	**Mission Planner**	**UGCS**	**MAVProxy**
Interfaz	Gráfica	Gráfica	Gráfica	Comandos
Coste	Gratuito	Gratuito	70$ al mes	Gratuito
MAVLink	Sí	No	Sí	Sí
Pilot PX4	Sí	Sí	No	Sí

Tabla 7.1: Comparación entre softwares de estación de drones[28]

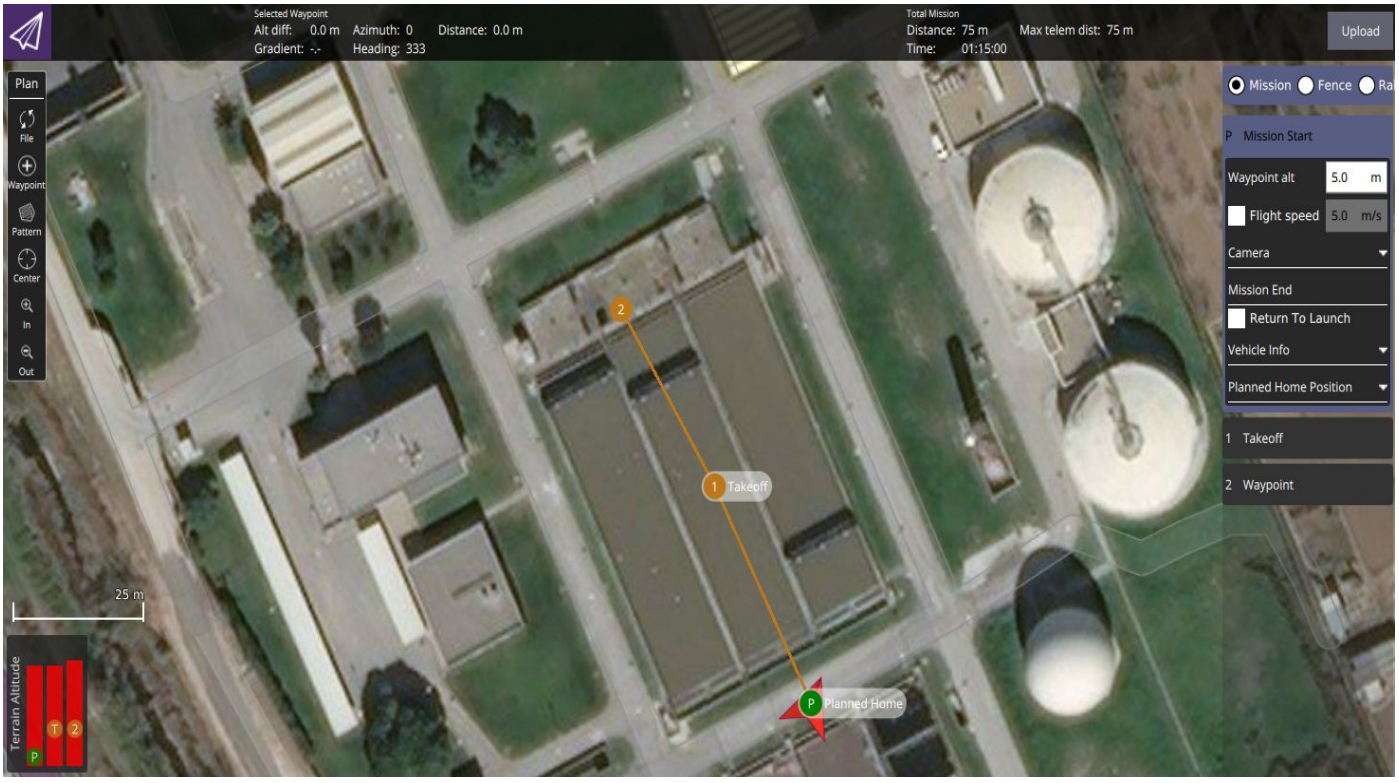

Figura 7.3: Visualización 2D del QGC.

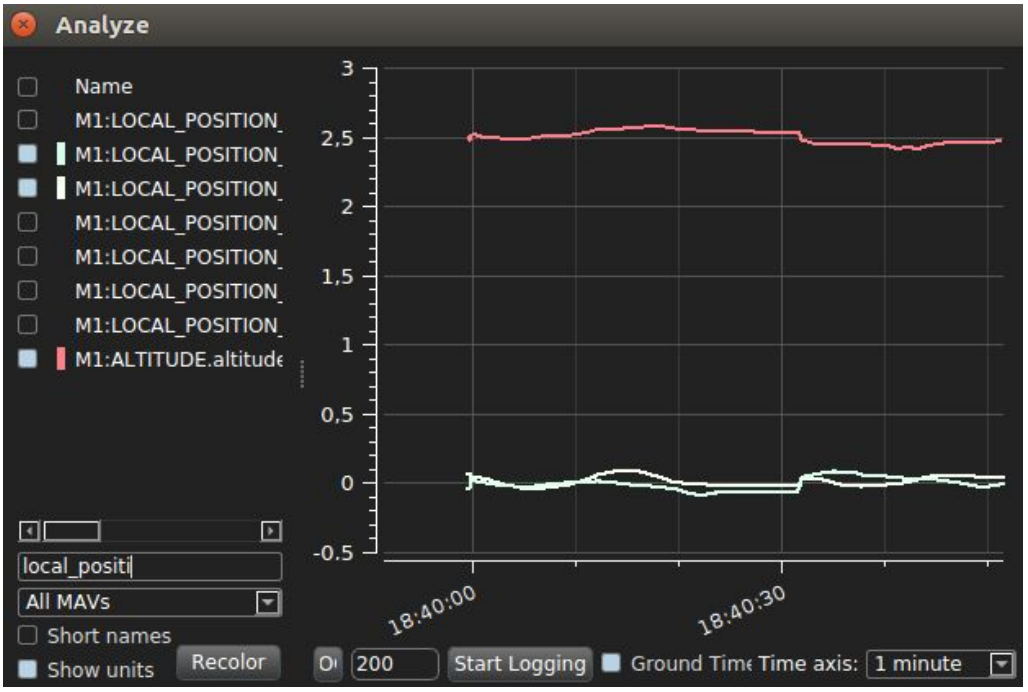

Figura 7.4: Widget Analyze. Se puede apreciar la posición local x,y y la altura.

7.3 Entorno de Simulación: Gazebo

A día de hoy existen una gran variedad de entornos de simulación para drones e incluso aviones no tripulados. Por lo tanto, se decidió recoger información de todos los entornos de simulación. Una vez obtenida la información se decidió hacer una criba basándose en unos criterios: la facilidad de usarse con el PX4, su coste, y si se podían utilizar junto a ROS. Obviamente también se tuvo en cuenta el requisito fundamental, el hecho de si el simulador podrá utilizar la técnica HITL.

Entorno	**Gazebo**	**Hector**	**RotorS**	**Xplane**	**jMavSim**	**AirSim**
ROS	Sí	Sí	Sí	No	Sí	Sí
HITL	Sí	No	No	Sí	Sí	Sí
Coste	Gratuito	Gratuito	Gratuito	No Gratuito	Gratuito	Gratuito
MAVLink	Sí	Sí	Sí	Sí	Sí	Sí
Facilidad con PX4	Sí	No	Sí	Sí	Sí	Nivel Medio
Sensores	Sí	Sí	Sí	Sí	No	Si

Tabla 7.2: Comparativa entre los entornos de simulación

Como se puede apreciar en la tabla 7.2, se descarto de inmediato tanto Héctor como RotorS[29], ambos nodos de ROS, que utilizan RVIZ como entorno de visualización. Debido a que se quería que fuera un producto open source y gratuito, se descarta el Xplane[30]. Por lo tanto solo quedan jMavSim[31], Gazebo y AirSim[32].

Todos estos entornos de simulación tienen guía y son aconsejadas por los creadores de PX4. Sin embargo, el simulador AirSim no está testeado en nuestra versión de Pixhawk, por lo tanto es complicado y arriesgado. Y el simulador jMavSim es demasiado simple para nuestros requisitos, ya que no se pueden añadir sensores y el entorno gráfico es pobre. Por lo tanto, nos decidimos quedarnos con Gazebo.

Gazebo es un simulador 3D de código abierto cinemático y dinámico para todo tipo de robots que permite recrear cualquier tipo de entorno complejo de forma realista, ya sea interior o exterior. En nuestro proyecto se utilizará la versión 7.14.

Sus principales características de este simulador son las siguientes:

- Contiene diversos plugins que nos permiten incorporar sensores al modelo del UAV y simularlos, como pueden ser cámaras de distintos tipos, GPS, IMU, láseres y lidars.
- Capacidad de simular modelos de robots propios, ya sea en su lenguaje de modelado de robots(SDF) o en el lenguaje de ROS, URDF, como se puede apreciar en la figura 6.5. Incluso nos permite cargar estos modelos en tiempo de ejecución.
- Nos permite la posibilidad de crear escenarios en ficheros de formato .world. En estos ficheros podemos añadir modelos SDF, variar la gravedad, el terreno y demás opciones.
- Simula de forma realista la física de los modelos, es decir, los cuerpos pueden chocar, coger, empujar, rodar o ponerse encima de otros modelos o del suelo. Para este propósito se utiliza la etiqueta *<colision>* y el motor de físicas ODE (Open Dynamics Engine)
- Gazebo cuenta con una gran comunidad de usuarios detrás y una gran cantidad de instituciones públicas y docentes creando contenido académico basados en este simulador.

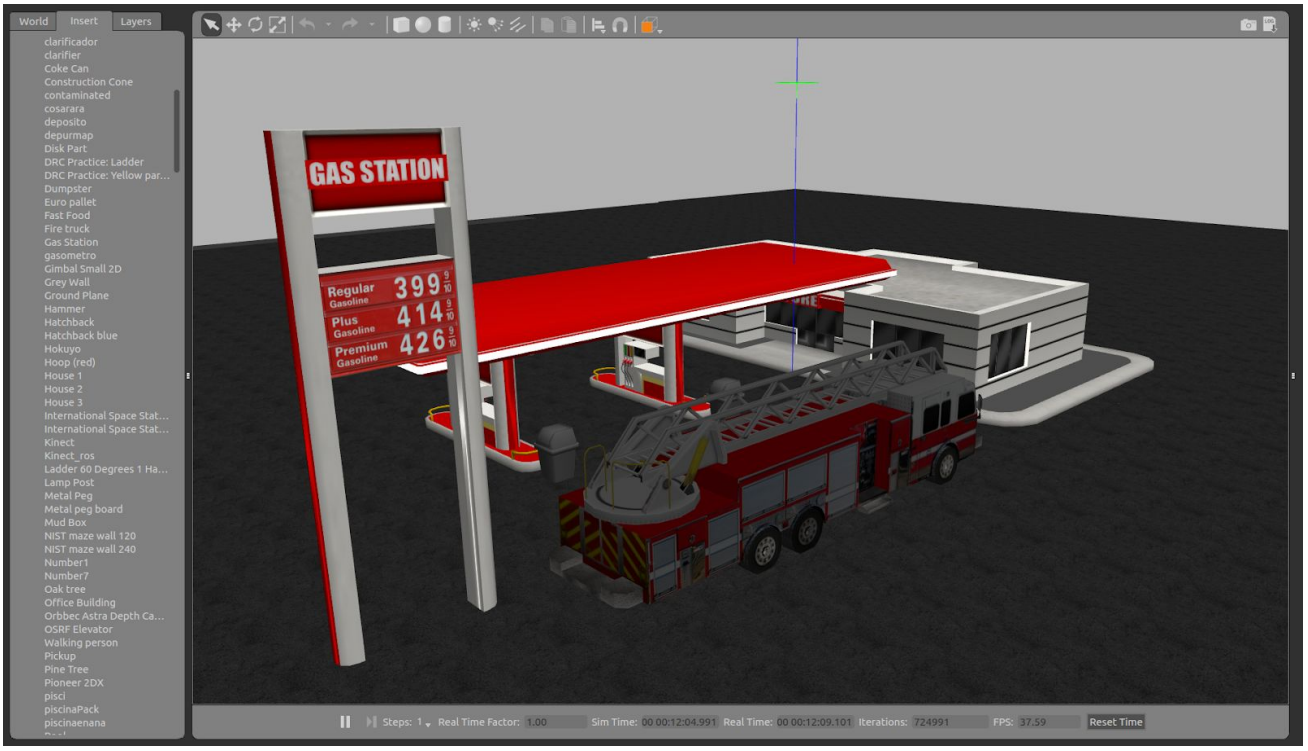

Figura 7.5: Visualización de una gasolinera y un camión de bomberos en Gazebo.

7.4 Pixhawk

Un controlador de vuelo o autopilot es un sistema que controla el vuelo del UAV sin la necesidad de la actuación humana. En nuestro caso nos centraremos en los autopilots orientados a quadcopter.

Pixhawk es una compañía independiente de hardware que crea controladores de vuelo de bajo coste, ideal para comunidades académicas y de aficionados. Esta controladora es ideal para los firmware autopilot PX4 y ArduPilot.

Como se ha especificado anteriormente, en este proyecto se ha trabajado con la controladora de vuelo Pixhack 2.8.4, una placa que utiliza el diseño hardware FMUv3. Esta placa posee las siguientes características:

- **Puerto I2C**, un bus de comunicación maestro-esclavo.
- 2 puertos de **Telemetría** y un puerto **GPS.**
- Un puerto **Futaba SBUS**, un puerto serial para conectar un dispositivo de control remoto de la marca Futaba.
- Entrada de señal **PPM** y de salida de **PWM**.
- Barómetro de alta precisión **MS5611**
- **LSM303D** acelerómetro / magnetómetro de 3 ejes y 14 bits.
- Unidad de **IMU** independiente, con una estructura de absorción de vibración.
- Puerto para tarjeta **Micro SD.**
- Procesador **STM32F427** de 2 mb con caché de 32 bits.
- Coprocesador de copia de seguridad **STM32F103** de 32 bits.
- Frecuencia básica de **168MHZ y 256K RAM**.
- **Tamaño** de 69 * 45 * 16 mm.
- **Peso**: 33g.

Figura 7.6: Placa Pixhack 2.8.4[33]

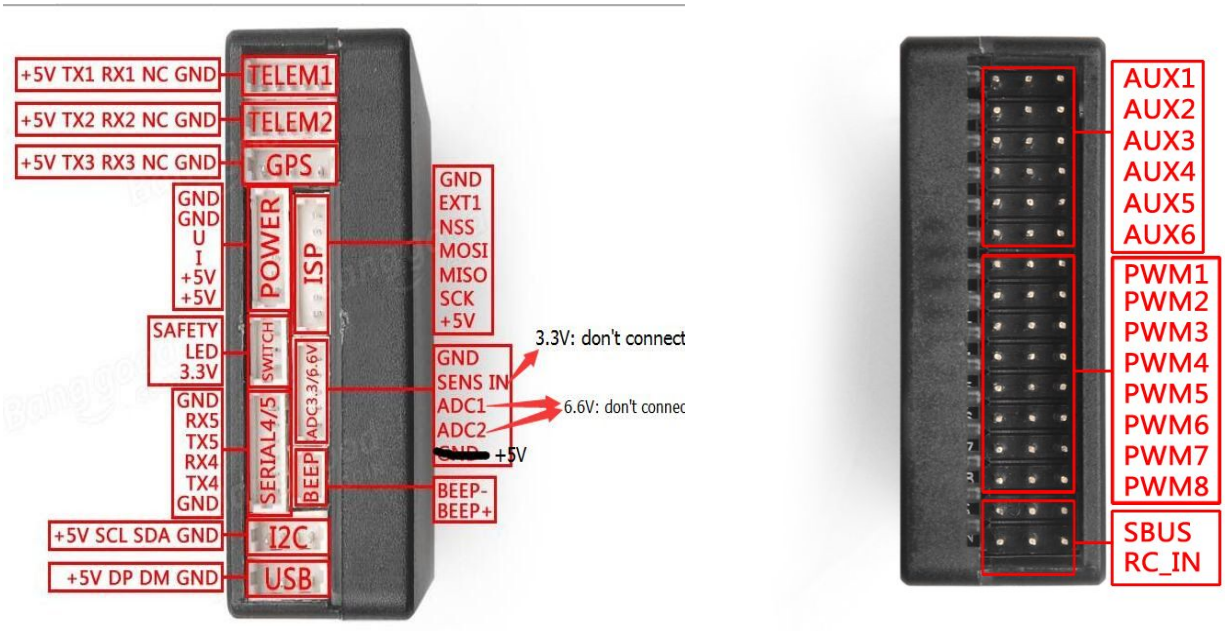

Figura 7.7: Pinouts de la Pixhack 2.8.4[33]

7.5 Firmware del autopilot

7.5.1 Estructura Firmware

Generalmente, las placas como la Pixhawk llevan instalado un firmware, que es el encargado de controlar el comportamiento del UAV. Su estructura está dividida en 3 capas, como se puede apreciar en la siguiente imagen.

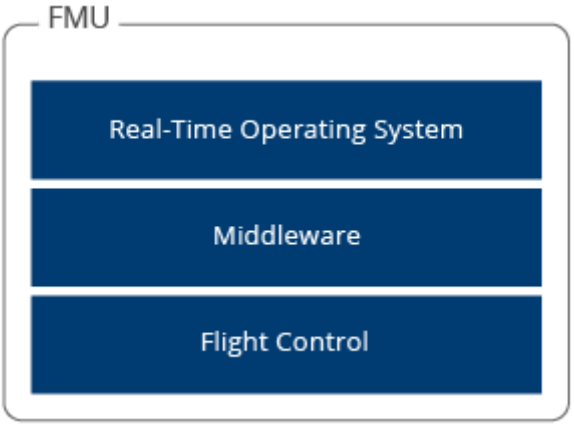

Figura 7.8: Estructura de un firmware moderno.

- **Capa RTOS**: Un controlador de vuelo necesita un sistema operativo de tiempo real, ya que nos aporta concurrencia y sobretodo por la necesidad de tener que completar ciertas tareas en un tiempo determinado, los cuales son un obligatorio para garantizar la seguridad y el control del UAV.
- **Middleware**: Es el software que se ejecuta entre el sistema operativo y las aplicaciones que se ejecutan en él y que permite el paso de información. En nuestro caso es una colección de librerías, drivers y herramientas que manejan información procedente de sensores y filtros de control.
- **Flight Control:** La parte más importante del controlador, ya que es la que se encarga de controlar todas las rutinas y comandos de control. Nos referimos a datos como la calibración, la odometría, telemetría y el control del motor.

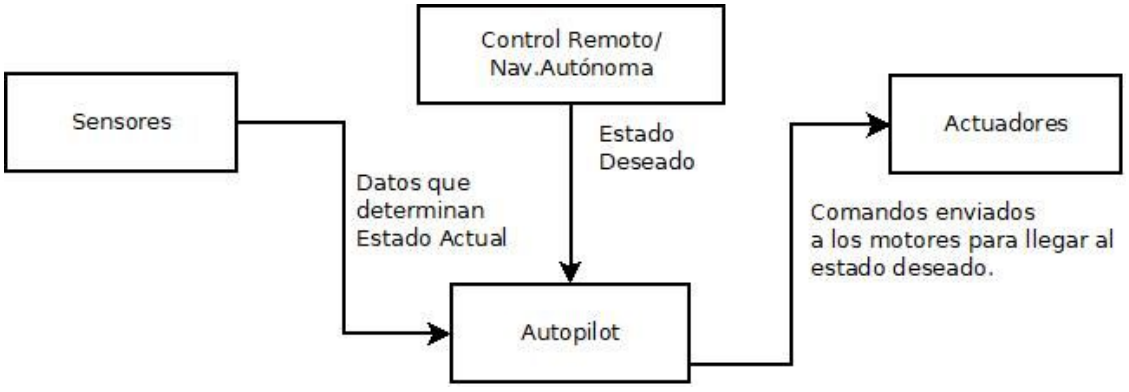

Figura 7.9: Diagrama del funcionamiento de un firmware.

Como se puede apreciar en la anterior figura, el objetivo principal de un autopilot es el de llegar al estado deseado. Este estado deseado se obtiene mediante una entrada, generalmente proveída de un sistema RC o bien de un nodo de control autónomo. El autopilot además también recibe información de los sensores, los cuales utiliza para generar el estado actual y una vez obtenido ambos datos, controlar el comportamiento de los actuadores para intentar alcanzar el estado deseado por el RC o el nodo.

7.5.2 PX4

El PX4 es el stack o firmware de controlador de vuelo muy versátil y ampliamente utilizado, creado por Dronecode, una plataforma de código abierto basada en drones, cuyo objetivo es la creación de una comunidad de desarrolladores que creen herramientas para el mundo de los drones.

La característica principal del PX4 es que es un firmware con una alta compatibilidad con múltiples sensores y vehículos, siendo capaz de controlar una grandísima cantidad de ellos. Obviamente, es compatible con el framework ROS y Gazebo y utiliza el RTOS NuttX, de código abierto.

La elección de este firmware viene decidida por el departamento, ya que es el firmware con el que decidieron trabajar y que ya tenían instalado en la Pixhawk. Aun así, si hubiera podido escoger, posiblemente hubiera efectuado la misma decisión. PX4 tiene como gran competidor a Ardupilot, también de código abierto. Sin embargo, la gran ventaja de PX4 respecto es que su proceso de desarrollo es continuo y bueno, ya que cuenta con actualizaciones cada pocos meses y un plan de futuro, mientras que Ardupilot ha reducido su frecuencia de actualización y de comunicación entre desarrolladores y comunidad.

También cabe destacar que en el terreno de la simulación, PX4 tiene una gran ventaja respecto Ardupilot, ya que sus herramientas para simulación son más maduras y confiables. Finalmente, decir que para el ámbito de investigación, PX4 es mejor debido a que es más personalizable.

7.6 RVIZ

Para la visualización de los sensores de forma visual, se ha optado por RVIZ. Como se mencionó anteriormente en el apartado 6.1.4, RVIZ es una herramienta de visualización de sensores y del entorno. Este programa ofrece distintos sistemas de visualización, permitiendo al programador añadir tipos de visualización que permiten visualizar el entorno percibido a través de los sensores del robot. Dentro de las posibilidades de visualización utilizaremos los siguientes tipos:

- **Axes:** Muestra un conjunto de ejes con coordenadas (x,y,z) en colores verde, rojo y azul.
- **Camera:** Crea un nuevo panel de renderizado superpuesto para la visualización de las cámaras del dron.
- **Image:** Muestra la visualización de lo que ve una cámara abriendo una nueva ventana.
- **Grid:** Muestra una rejilla centrada en el origen.
- **Laser Scan:** Muestra los datos recibidos de un scanner láser, con diferentes opciones de renderizado y acumulación de datos.
- **Marker:** Permite mostrar formas primitivas como figuras geométricas a través de topics de posición.
- **Odometry:** Muestra las posiciones estimadas por la odometría con el paso del tiempo.
- **PointCloud2:** Muestra los datos recibidos de la nube de puntos con diferentes opciones de renderizado o acumulación.
- **Range:** Muestra conos que representan la información recibida del sonar o de los sensores IR de rango.
- **RobotModel:** Visualización del robot en 3D.

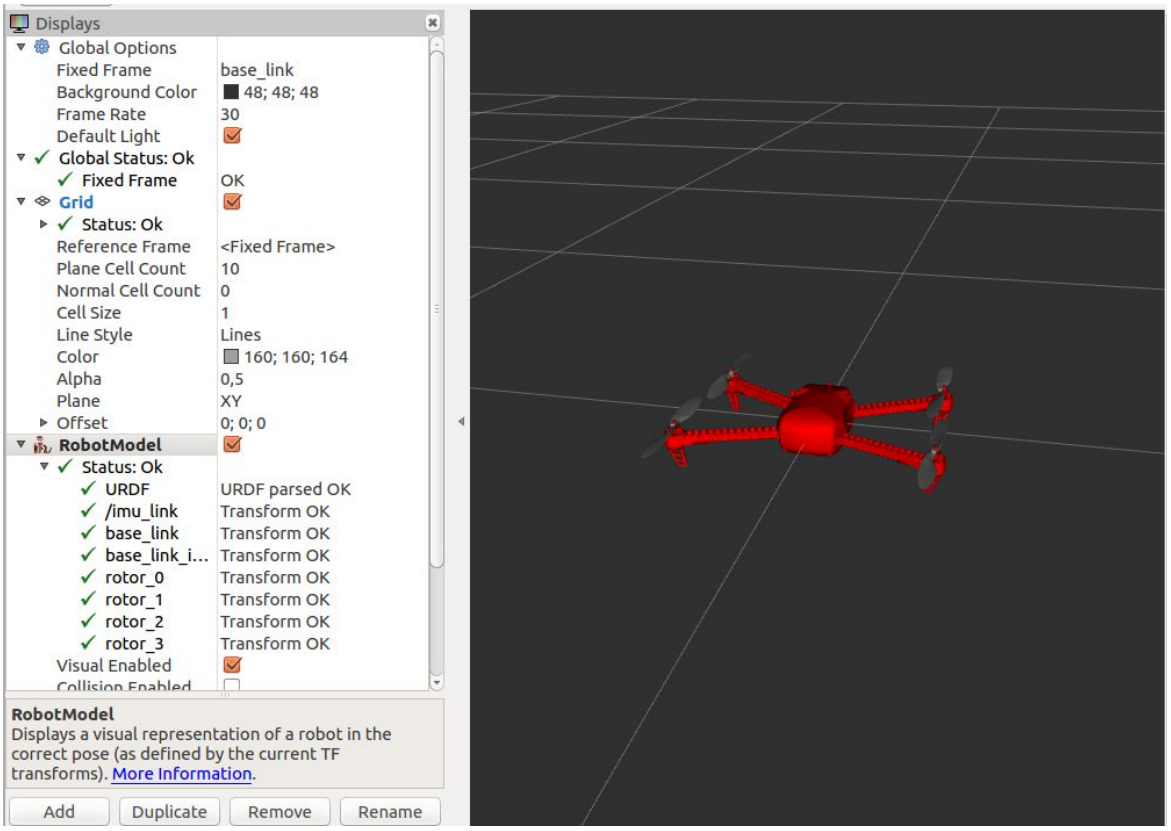

Figura 7.10: RobotModel del quadcopter Iris en RVIZ

7.7 Análisis de logs: Flight Review

Como hemos mencionado anteriormente, un log es un fichero que registra gran parte de los acontecimientos del que afectan a un proceso o sistema. Este tipo de archivo es realmente importante, ya que nos permite analizar los datos del UAV durante el vuelo y en caso de error, descubrir los motivos.

La generación de logs se produce automáticamente en el QGroundControl y podemos definir qué tramos abarcar en este fichero mediante el parámetro SDLOG_MODE.

Una vez obtenido el log, necesitamos una plataforma que pueda analizarlos. Para ello se ha optado por Flight Review[30], una herramienta online. Al ser una web, permite al usuario ver en cualquier instante y dispositivo el análisis y facilita poder compartirlo entre diversas personas.

Para crear el reporte, solo tendremos que enviar el fichero log y el email, para que cuando el análisis se acabe de procesar, se nos envíe la dirección. Igualmente, también se puede añadir una descripción del vehículo, del test e incluso un video del vuelo.

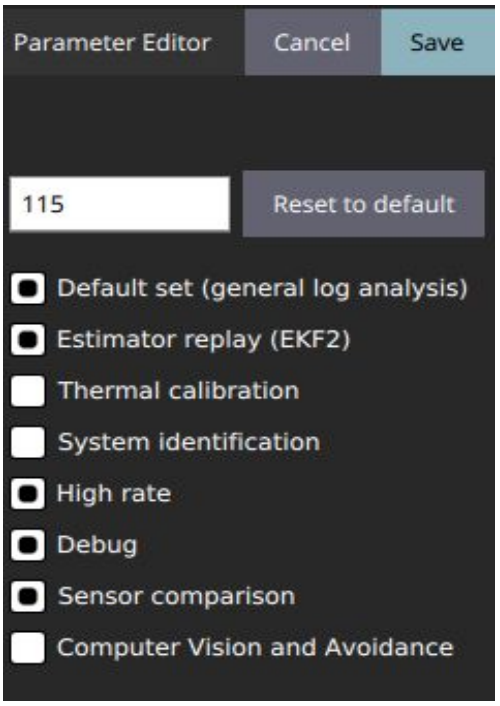

Figura 7.11: Modificación del parámetro SDLOG_MODE

Dentro del informe podemos apreciar gráficos de distintos campos, como la posición local y global del drone durante la simulación, valores de la IMU, vibración, velocidad y valores de sensores. A continuación, en la figura 6.13 se ofrece una imagen de ejemplo de un trayecto en círculos del dron.

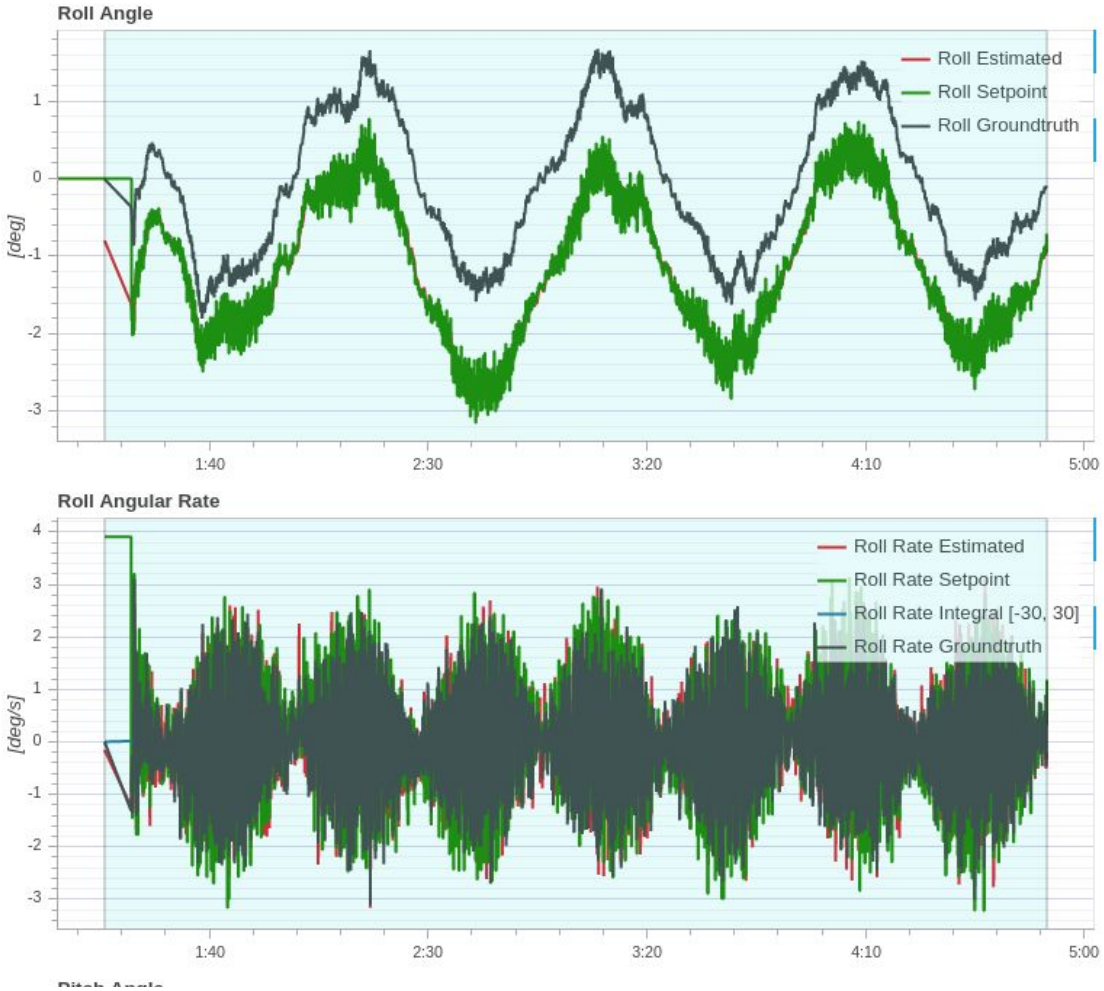

Figura 7.12: Roll Angular de un trayecto reportado en Flight Review

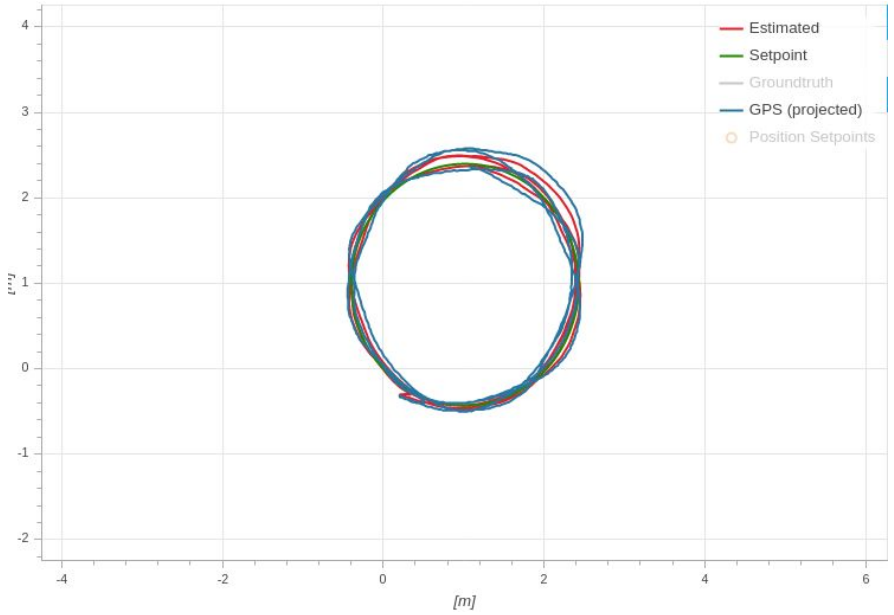

Figura 7.13: Trayectoria del vuelo.

8.Creación del Simulador

En este apartado de la memoria se detalla el simulador. Se explica el circuito que lo compone, la visualización de datos, su configuración y sus conexiones. Asimismo, también se explicará cómo conectar el sistema R/C.

8.1 Configuración RC

Antes de la configuración del HITL, se decidió utilizar un mando RC para poder controlar el dron simulado de forma realista y aproximada a la realidad. Para ello se utiliza el RC Futaba T8J con el receptor R2008SB.

El siguiente esquema muestra la conexión del receptor con la Pixhawk.

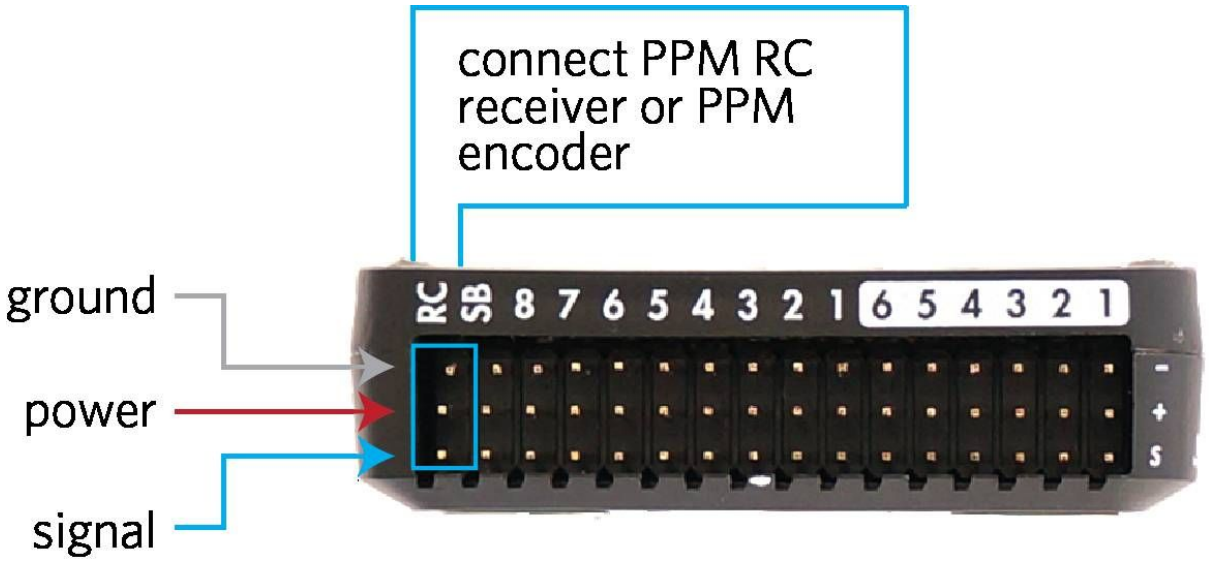

Figura 8.1: Conexión entre el receptor y la Pixhack[33].

Una vez conectada la Pixhawk al receptor, se tiene que configurar el sistema de radio control. Para ello, se conecta la Pixhawk al PC via serial y se abre el QGC, es decir el software de estación de tierra de drones, ya que es este programa el que nos permite calibrar y configurar el radio control.

Se optó por una configuración de un solo canal, utilizando el canal 5 para el cambio entre modo de vuelo.

Figura 8.2: Futaba T8J[21]

El eje de la izquierda se usará para el movimiento del eje Z. El movimiento vertical será para controlar la altura, mientras que el movimiento horizontal para controlar la rotación roll. Por su parte, el eje de la derecha se usará para controlar el movimiento x e y del dron.

Mediante el canal 5, el primer switch superior de la izquierda se cambia entre 3 modos de vuelo. Estos serán el modo Land, Altitude y Mission. Con el canal 6, el segundo switch superior de la izquierda, se ejecuta el kill switch, que desconecta los motores del dron, permitiendo un aterrizaje forzoso. Finalmente, el canal 7, el primer switch superior de la derecha se usará para armar y desarmar el dron.

8.2 Configuración Simulador

Una vez presentados todos los elementos y tecnologías utilizadas en este proyecto, pasamos a ver la integración de las mismas en el simulador.

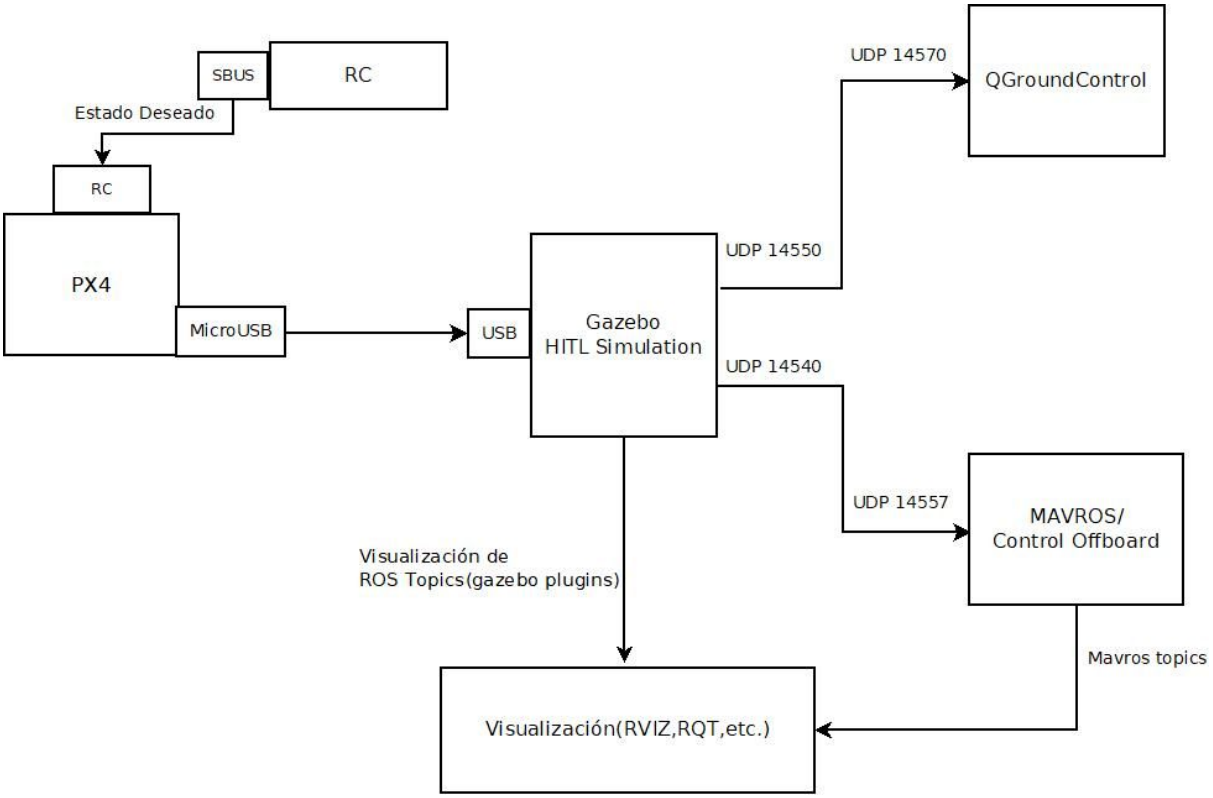

Figura 8.3: Circuito del simulador HITL.

El diagrama de arriba muestra el entorno de simulación HITL. La comunicación mediante el simulador Gazebo y el PX4 se efectúa mediante comunicación serial USB. Gazebo proporciona información de los sensores a la Pixhawk y este le devuelve los datos de los actuadores, que son los que mueven al dron simulado en Gazebo. El paquete Mavros nos permite ver la información del dron simulado, es decir su estado actual, mediante ROS Topics, ya que transforma el protocolo Mavlink a topics de ROS. También nos permite publicar en estos topics y por lo tanto, alterar el estado deseado y transmitirselo a la PX4 vía Gazebo.

QGC tiene una función similar a Mavros en este simulador. Gracias a su interfaz gráfica, podemos efectuar acciones como cambiar el modo de vuelo del PX4, editar la lista de misiones o *waypoints* y ejecutarla. Las misiones son coordenadas GPS en las cuales el dron tiene que ir y efectuar una acción(pasar, aterrizar, despejar). También tenemos la misión volver a home, el cual es una coordenada que podemos editar.

En resumen, actúa como Mission Planner. Evidentemente, también se puede ver los datos Mavros mediante la pestaña *Analyze* y ver como el dron se mueve en un mapa similar al de Google Maps.

Las herramientas de visualización de ROS se ejecutarán para ver principalmente los sensores visuales, tales como cámaras, láseres o topics no incluidos en Mavros.
Finalmente, el RC se encarga de mover de forma manual el dron mediante los aceleradores y cambiar el modo del dron de forma manual.

Para la ejecución de este simulador se tienen que ejecutar los siguientes pasos:

1. Conectamos la placa al ordenador con el USB. Abrimos el QGC.
2. Abrimos la pestaña *Parameters* y buscamos el parámetro SYS_HIL. Lo ponemos a valor 1. Acto seguido, vamos a *Airframe* y seleccionamos el HIL quadcopter X. Otra forma de ejecutar este paso es cambiar el parámetro SYS_AUTOSTART a 1001. Si se desea simular un quadcopter con gimbal, el valor del parámetro debe ser 4002. Pulsamos la opción Apply y Restart desde la pestaña de Airframe o bien desde Parameters hacemos un reboot de la placa.
3. Nos aseguramos que el UDP está configurado en la sección *General*.
4. Una vez efectuado estos pasos, cerramos el QGC y desconectamos el USB.
5. Abrimos el terminal y vamos al directorio donde tenemos el firmware. Construimos PX4 con el Gazebo.

```
cd <Firmware_clone>
DONT_RUN=1 make px4_sitl_default gazebo
```

6. Abrimos el modelo iris.sdf y en la sección mavlink_interface, cambiamos los valores de SerialEnabled y hil_mode a true. También cambiar el valor de SerialDevice según la conexión de nuestro PC. Para ello se recomienda usar `ls /dev`, que nos dará todas las conexiones.

```
<plugin name='mavlink_interface' filename='librotors_gazebo_mavlink_interface.so'>
  <robotNamespace/>
  <imuSubTopic>/imu</imuSubTopic>
  <imu_rate>170</imu_rate>
  <gpsSubTopic>/gps</gpsSubTopic>
  <mavlink_addr>INADDR_ANY</mavlink_addr>
  <mavlink_udp_port>14560</mavlink_udp_port>
  <serialEnabled>true</serialEnabled>
  <serialDevice>/dev/ttyACM0</serialDevice>
  <baudRate>921600</baudRate>
  <qgc_addr>INADDR_ANY</qgc_addr>
  <qgc_udp_port>14550</qgc_udp_port>
  <hil_mode>true</hil_mode>
  <hil_state_level>false</hil_state_level>
```

7. Ejecutamos los siguientes comandos, los cuales son necesarios para lanzar gazebo junto a ROS.

```
source ~/catkin_ws/devel/setup.bash    # (optional)
source Tools/setup_gazebo.bash $(pwd)
$(pwd)/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
```

8. Finalmente, lanzamos el HITL. Cuando la placa efectúe un sonido, significa que todo estará correcto y por lo tanto, podremos efectuar las conexiones UDP. La del QGC se efectúa sola, por lo tanto solo hace falta abrirlo.

```
roslaunch gazebo_ros empty_world.launch
world_name:=$(pwd)/Tools/sitl_gazebo/worlds/iris.world
```

9. La conexión y ejecución de Mavros se efectúa mediante el siguiente comando:

```
roslaunch mavros px4.launch
fcu_url:="udp://:14540@192.168.1.36:14557"
```

10. Para usar las herramientas de visualización, se ejecuta el comando rosrun + el nombre de la herramienta. Por ejemplo para rviz

```
rosrun rviz rviz
```

9.Modelado del Dron y del Entorno.

En esta sección se detalla como se ha modelado la aeronave y como se han añadido sensores y sus pruebas. También se explicará el lenguaje SDF de Gazebo y la creación de la EDAR.

9.1 SDF y URDF

Como se ha mencionado anteriormente, SDF[34] es el lenguaje de modelado de objetos y robots para Gazebo. Sin embargo cabe destacar que el lenguaje de modelado de robots que utiliza ROS es URDF[35]. Sin embargo, la diferencia entre estos dos formatos son mínimas, ya que ambos se basan en el lenguaje de etiquetas.

En este proyecto, se ha decidido dar un mayor enfoque al formato SDF, ya que SDF es mejor para crear mundos simulados, ya que se pretende crear un entorno similar al de la EDAR de Sant Feliu de Llobregat, en un fichero .world. Además, el firmware PX4 da un mayor apoyo y uso al formato SDF.

9.2 Características SDF

Como hemos indicado, SDF es un lenguaje de etiquetas por lo que existe una estructura en árbol donde existen etiquetas con una serie de atributos. En esta explicación, indicaremos lo más básico y utilizado en la recreación de la depuradora.

9.2.1 Model

Para la creación de un objeto, se utiliza la etiqueta <model>, la cual alberga todos las partes físicas del objeto, sus uniones entre sí y sus funcionalidades.A continuación se explican sus principales hijos.

- **Link:** La etiqueta Link representa un cuerpo rígido nombrado, es decir es un componente físico visible en Gazebo. Sus principales etiquetas son:
    - **Colisión**: elemento necesario para la representación del elemento en gazebo, ya que dota al elemento de capacidad de percepción de colisiones.
    - **Visual:** Define la forma del elemento mediante el atributo geometry. Principalmente, podemos crear formas cilíndricas, esféricas o cuadradas/rectangulares. Sin embargo, se puede linkear a un archivo Collada(.dae), el cual es un objeto más desarrollada. Este tipo de ficheros se puede crear mediante Blender.
    - **Inertial**: Definición de las características inerciales del elemento.

- **Joint:** Esta etiqueta representa la unión cinemática entre dos elementos, el cual uno ejerce de hijo y el otro de padre. Existen diversos tipos de uniones según el grado de libertad entre los elementos, pero nos centraremos simplemente en dos:
    - **Revolute**: La articulación gira alrededor de un eje entre ciertas cotas definidas.
    - **Fixed**: 0 grados de libertad.

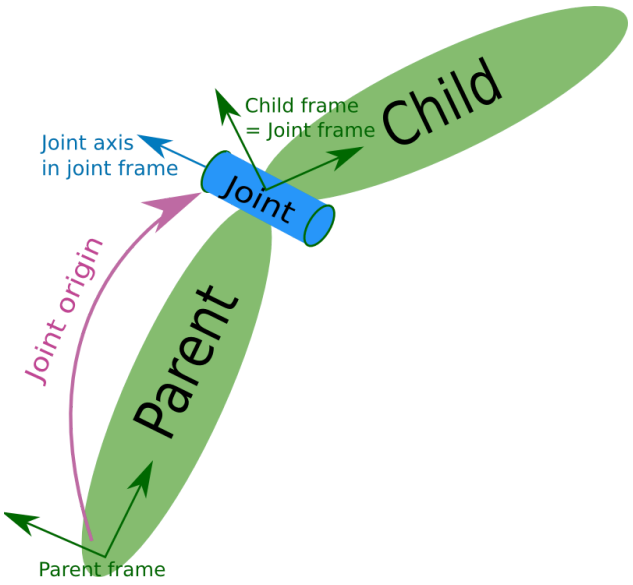

Figura 9.1: Explicación de la etiqueta Joint[36].

- **Plugin:** En esta sección se asocia un plugin al modelo o bien a un cierto elemento link de éste. Además, también se puede modificar atributos del plugin. En la siguiente imagen tenemos un ejemplo de plugin.

```
<plugin name='rosbag' filename='libgazebo_multirotor_base_plugin.so'>
  <robotNamespace/>
  <linkName>base_link</linkName>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
</plugin>
```

Figura 9.2: Plugin rosbag

9.2.2 Config

Para poder utilizar un fichero con este formato, aparte del modelo hace falta crear un archivo de configuración del fichero. Este fichero de metadatos contiene una estructura similar a la siguiente.

```
<?xml version="1.0" ?>
<model>
    <name>cosarara</name>
    <version>1.0</version>
    <sdf version="1.6">model.sdf</sdf>
    <author>
        <name></name>
        <email></email>
    </author>
    <description></description>
</model>
```

Todos estas etiquetas son de carácter obligatorio, ya que deben tener valores. A continuación se especifica su significado:

- **name:** El nombre del modelo.
- **version:** Indicar la versión del modelo y enlazarlo con el archivo.
- **author:** Nombre y email del creador del modelo.
- **description:** Descripción del modelo. Debe incluir en qué consiste o qué es el modelo y listar los plugins que utiliza.

9.3 Creación EDAR

Para la recreación de la estación depuradora de St.Feliu de Llobregat, se ha tenido que modelar y crear diversos modelos y objetos de distintas características.

Figura 9.3: EDAR St.Feliu de Llobregat[37]

Como se puede apreciar en la imagen, la estación depuradora está compuesta por un gasómetro, 2 reactores, clarificadores, dos edificios que elevan el agua mediante tornillos de arquímedes y decantadores. Para la recreación de los edificios no se ha podido disponer de los planos detallados, sin embargo se ha podido disponer de información concerniente a la altura, anchura y profundidad de los elementos anteriormente nombrados.

Para la construcción del terreno, se utilizó una imagen en escala de grises del terreno, mediante el uso de la web terrain.party. Se exporta a .png y se puede utilizar como terreno para nuestro mundo en Gazebo. Esto es realmente útil porque de esta forma se puede simular el terreno de cualquier lugar del mundo de forma realista y sencilla.

Para crear la carretera, se puede utilizar la etiqueta road, una etiqueta que crea una carretera de punto a punto del mapa. También se puede indicar su anchura y su altura respecto a la coordenada z = 0. En la figura 8.4 se puede apreciar un ejemplo.

```
<road name="my_road11"> <!--recta pequeña-->
  <width>6.0</width>
  <point>-137.4 88.91 0.1</point>
  <point>-137.4 27 0.1</point>
</road>
```

Figura 9.4:Creación de una carretera

Respecto a la construcción de los otros elementos, se decidió hacer uso de la herramienta Model Editor de Gazebo. Esta herramienta nos permite crear modelos simples, enfocándonos principalmente en modificar y crear con esta herramienta las etiquetas <link> y los joints entre links. Después de esto, a la gran mayoría de modelos creados se les modifica o añaden atributos, como <static>, <gravity> para refinar su comportamiento.

Finalmente, para que los objetos sean visualmente más realistas, se puede adjuntar un mesh mediante un archivo Collada o bien se les puede asignar texturas mediante un fichero .png a cada elemento link del modelo, todo esto en la etiqueta <visual>. La estructura de un modelo en el cual utilizamos ambas técnicas es similar a la siguiente imagen

```
esaii-admin@esaii-omen:~/.gazebo/models/pine_tree$ tree
.
├── materials
│   ├── scripts
│   │   └── pine_tree.material
│   └── textures
│       ├── bark_diffuse.png
│       └── branch_2_diffuse.png
├── meshes
│   └── pine_tree.dae
├── model.config
└── model.sdf
```

Figura 9.5: Estructura de un modelo .sdf

Para la creación de meshs utilizaremos Blender, un programa gráfico que permite la creación de objetos 3D y para linkear el modelo con el mesh, se retoca la etiqueta <geometry> de <visual> de la siguiente manera:

```
<visual name="branch">
      <geometry>
        <mesh>
          <uri>model://pine_tree/meshes/pine_tree.dae</uri>
        </mesh>
      </geometry>
```

Si solamente se quiere modificar la textura del modelo, tenemos que crear un fichero .material donde enlazaremos con la textura y modificar esta textura mediante el lenguaje OGRE.

```
material Piscina
{
  receive_shadows off
  technique
  {
    pass
    {
      texture_unit
      {
        texture water.png
        filtering anistropic
        max_anisotropy 16
      }

    }
  }
}
```

Al final, tras crear todos los modelos de la depuradora y poner los modelos en la posición ideal, tenemos la EDAR simulada.

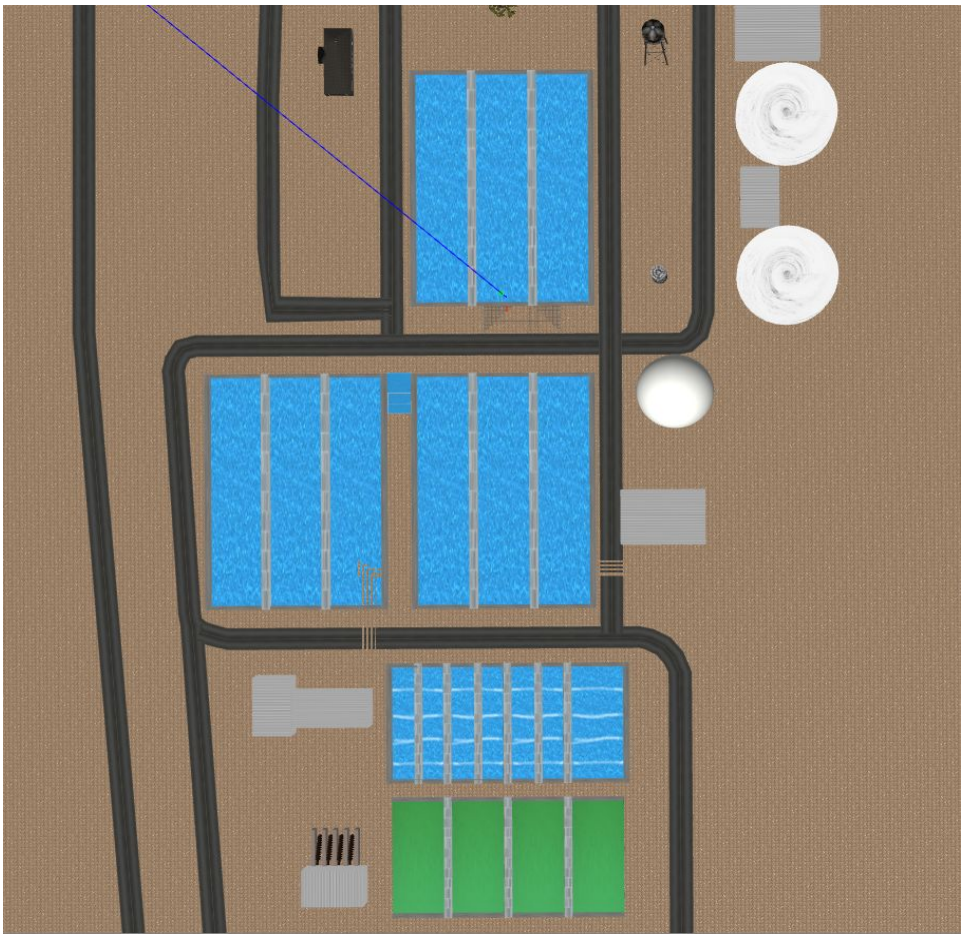

Figura 9.6: Simulación de la EDAR

Figura 9.7: Simulación de la EDAR(2).

9.3 Modelado del Drone

En esta sección se explica como esta creado el modelo de quadcopter generado por el firmware PX4 y como modificarlo para crear un modelo personalizado.

9.3.1 Modelo Iris.sdf

El firmware PX4 nos proporciona un modelo de quadcopter básico, el modelo iris.sdf el cual tiene la base y los 4 motores y todos los plugins necesarios para su funcionamiento, tales como el plugin de los motores, interficie MAVLINK, plugin de la IMU y el plugin de GPS. Estos plugins tiene atributos los cuales se pueden modificar.También nos proporciona los mesh y texturas para que visualmente se vea como un dron.

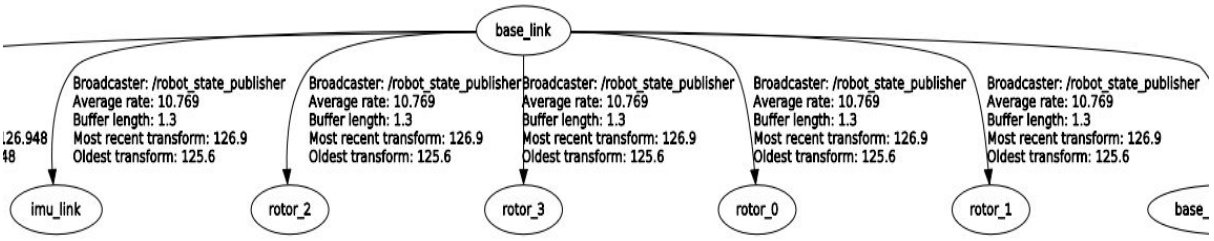

Figura 9.8: Estructura del quadcopter Iris

Como podemos apreciar, la estructura del dron consiste en una base, que hace de chasis, unida a 4 rotores, que son elemento motor del vehículo. También contamos con imu_link, elemento al cual está conectado el plugin de la imu. El código del iris.sdf es visible en el anexo 3.

9.3.2 Creación de un modelo Iris personalizado

Una vez entendido el funcionamiento de los plugins y los elementos físicos del modelo de quadcopter proporcionado por el firmware, pasamos a la etapa de añadirle los sensores que no están en el modelo iris.sdf, como cámaras, lidar, lasers y optical flow.

Para ello, crearemos un nuevo modelo .sdf a partir del modelo iris.sdf proporcionado, para acto seguido unirlo a los modelos de los sensores mediante un joint, generalmente de tipo revolute o fixed. En el siguiente código vemos un ejemplo, donde se une una camara y el modelo estándar mediante un joint de tipo fixed.

```
<?xml version='1.0'?>
<sdf version='1.5'>
  <model name='iris_custom'>

    <include>
      <uri>model://iris</uri>
    </include>

       <!-- Elemento de la camara -->
      <link name="camera_link">
      <pose>0.0 0.0 -0.05 0 0 0</pose>
      <inertial>
        <mass>0.1</mass>
      </inertial>
      <visual name="visual">
        <geometry>
          <box>
            <size>0.2 0.2 0.2</size>
          </box>
        </geometry>
      </visual>
      <!-- Def y configuración del sensor -->
      <sensor name="camera" type="camera">
        <camera>
          <horizontal_fov>1.047</horizontal_fov>
          <image>
            <width>320</width>
            <height>240</height>
          </image>
          <clip>
            <near>0.1</near>
            <far>100</far>
          </clip>
        </camera>
        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
      </sensor>
    </link>

    <joint name="camera_joint" type="fixed">
```

```
      <child>camera_link</child>
      <parent>iris::base_link</parent>
      <axis>
        <xyz>0 0 1</xyz>
        <limit>
          <upper>0</upper>
          <lower>0</lower>
        </limit>
      </axis>
    </joint>

  </model>
</sdf>
```

Una vez creado los ficheros .sdf y .config, creamos una carpeta con el nombre del modelo(en nuestro caso, nombraremos a este modelo iris_custom) y la ubicamos en el path /../Firmware/Tools/sitl_gazebo/models.

Para ejecutar este modelo en SITL para comprobar que es correcto, ejecutamos la siguientes comandas por la terminal:

```
cd <Firmware_clone>
DONT_RUN=1 make px4_sitl_default gazebo
source Tools/setup_gazebo.bash $(pwd)
$(pwd)/build/px4_sitl_default
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)
export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/Tools/sitl_gazebo
roslaunch px4 posix_sitl.launch vehicle:= iris_custom
```

El modelo que se verá debería ser el siguiente.

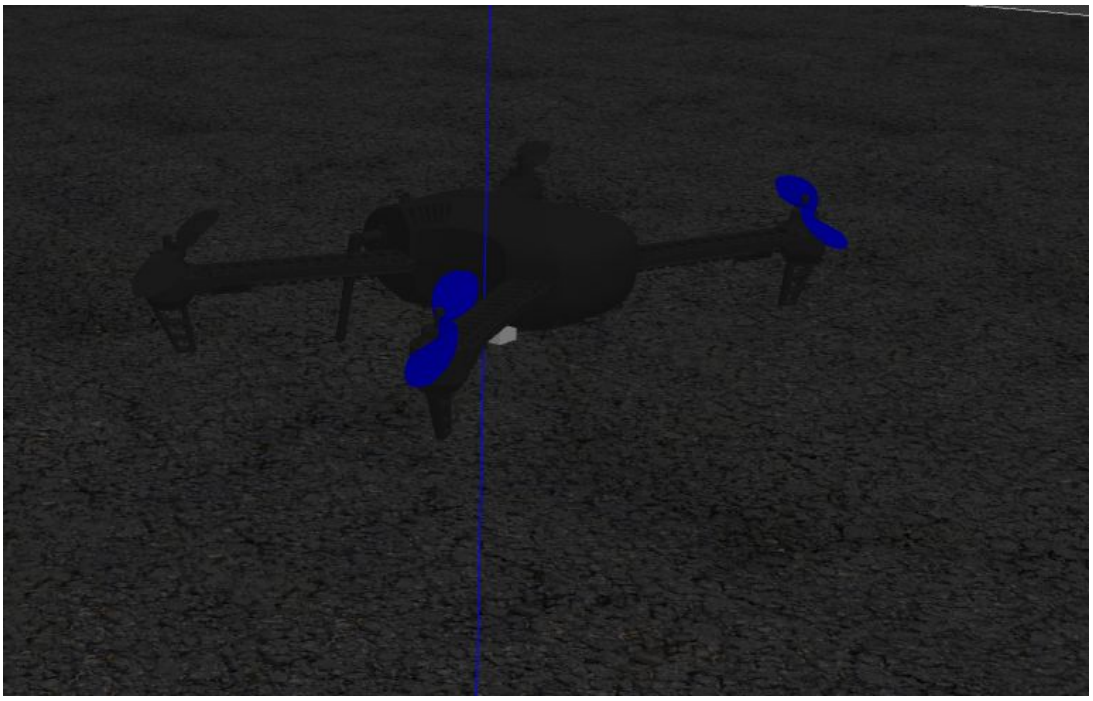

Figura 9.9: Dron iris con cámara.

10.Pruebas de Sensores

En este apartado se efectuaran distintas pruebas para comprobar el funcionamiento de los sensores aplicados al dron. Los sensores que se testean serán los siguientes:

- **Rangefinder**
- **Camara FPV**
- **Láser 2D**

10.1 Prueba RangeFinder

Un rangefinder es un aparato que mide la distancia entre un aparato y un objetivo. En este caso, se usa un rangefinder láser, es decir, se utiliza un rayo láser para determinar la distancia.

Para la primera prueba, se puso el UAV en la posición (0,0,3) para poder comprobar si el valor del rangefinder era de 3.

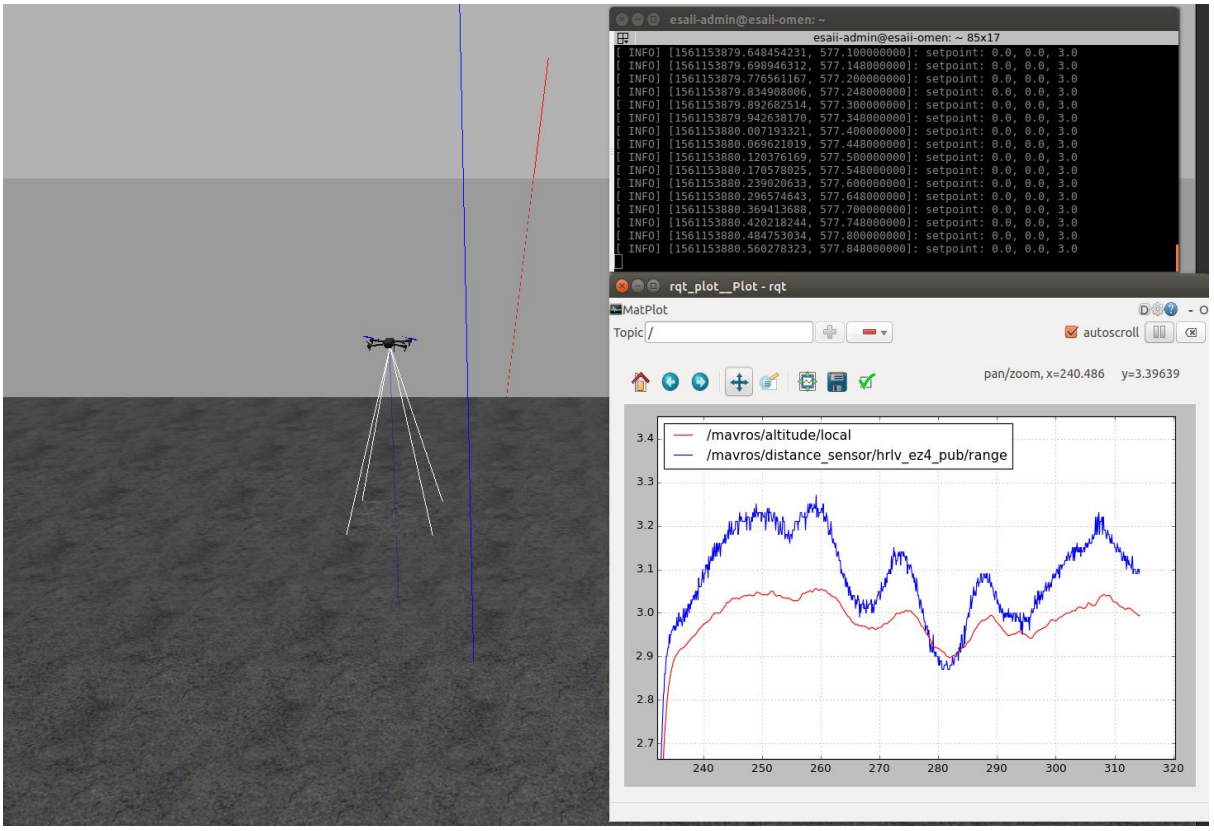

Figura 10.1: Dron con Rangefinder y gráfica de posición

Como se puede apreciar en la gráfica anterior, el valor que da el drone es de aproximadamente 3, aceptable dado si consideramos que se simula tanto cinemática como la dinámica y por lo tanto el UAV oscila y se balancea.

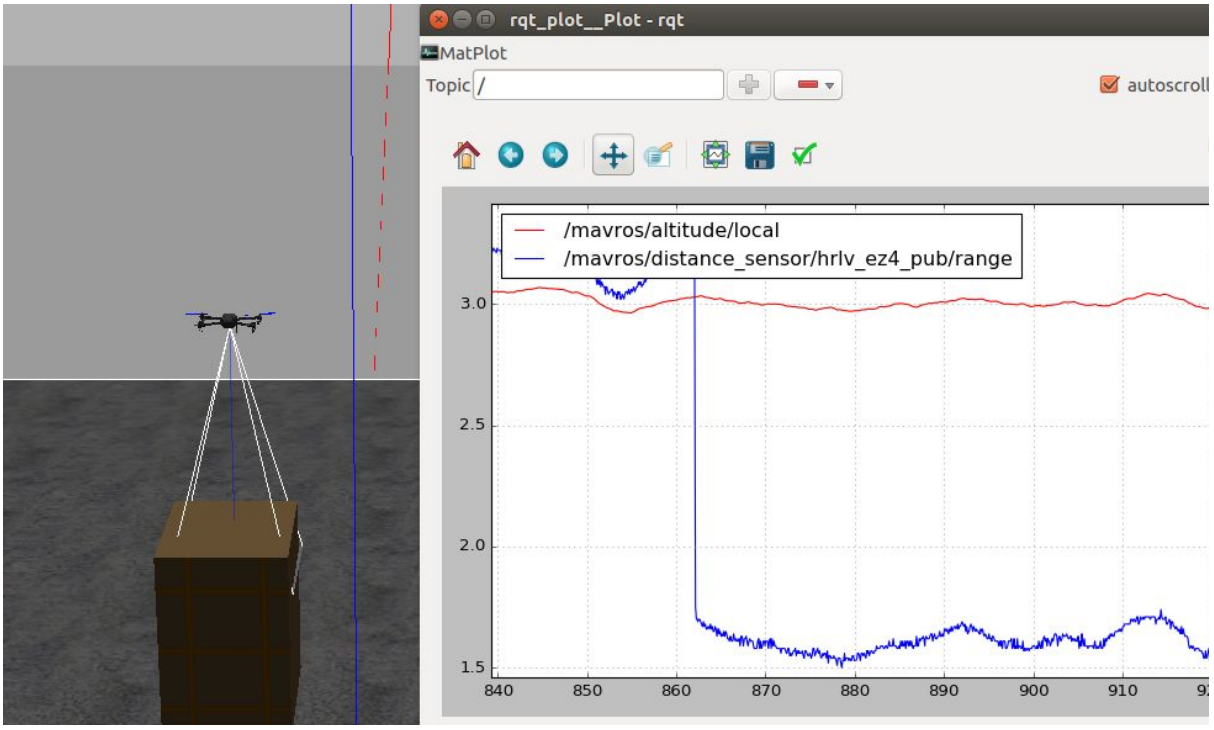

Figura 10.2: Dron iris con láser apuntando a caja y gráfica con valor de altura y laser.

Acto seguido, ponemos un objeto entre el dron y el suelo, para ver si el valor del rangefinder varia. La altura de la caja es de 1.4 metros. Como podemos apreciar en la gráfica, el valor del rangefinder baja hasta quedarse en los 1.6 metros aproximadamente.

Para comprobar que el valor producido por el láser es directo/sin procesar, decidimos mover el dron a gran velocidad. Esto provoca que el dron se incline y por lo tanto, que el rayo del láser sea diagonal. Como podemos apreciar en la siguiente imagen, el valor del láser se dispara al moverse el drone. Destacamos que la bajada producida en el instante 45 es debido a un cambio de dirección del drone.

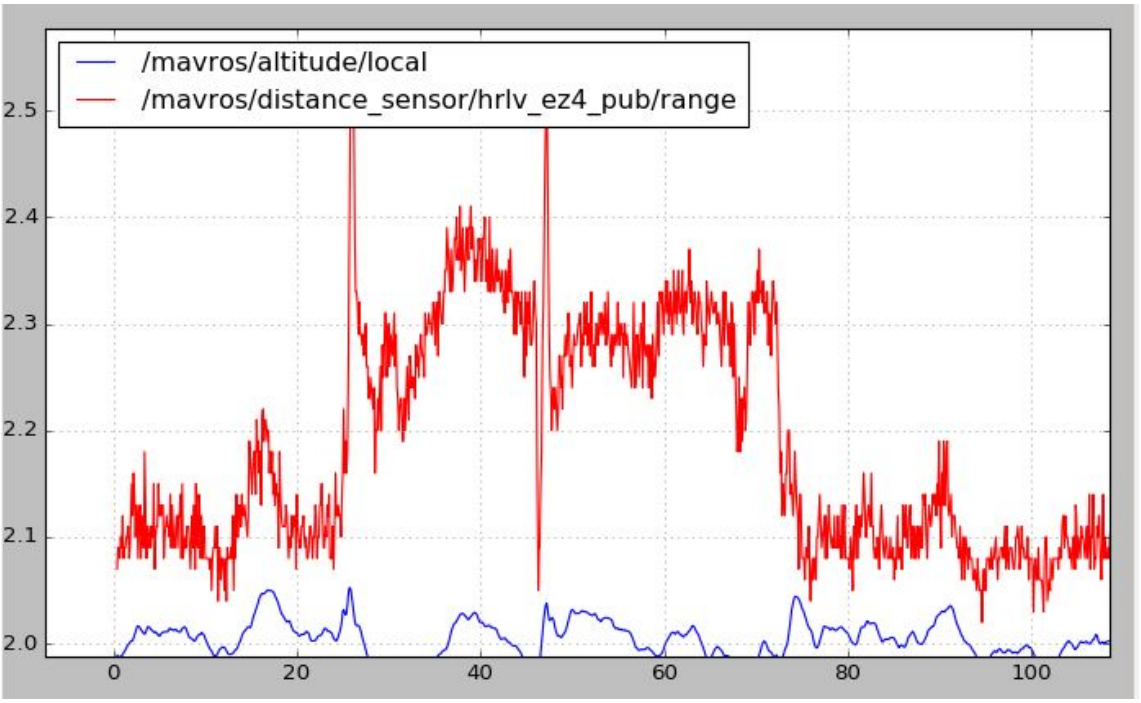

Figura 10.3: Gráfica comparativa entre altura y valor del láser al moverse el dron.

Como se ha podido apreciar en las pruebas, el rangefinder tiene un margen de pequeño,pero requiere de procesado con información adicional, como en la vida real.

9.2 Prueba con cámara fpv

A continuación se harán pruebas con una cámara fpv. Es una cámara que nos permite visualizar lo que ve el dron en primera persona.

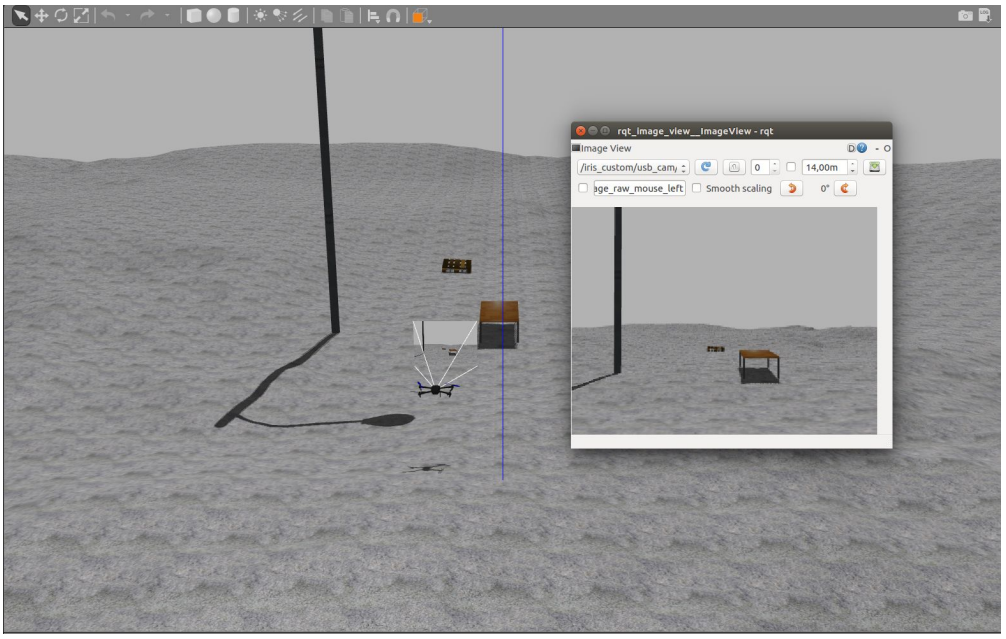

Figura 10.4: Gazebo y visualización de la cámara del dron.

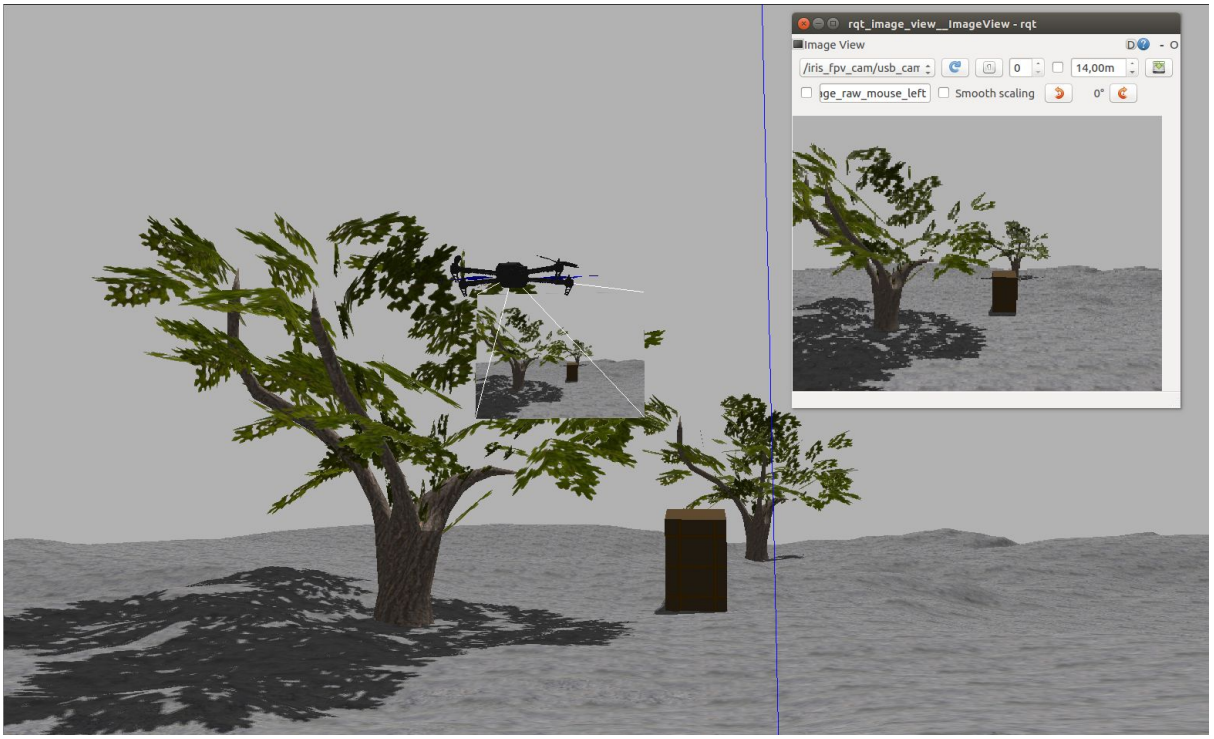

Figura 10.5: Gazebo e imagen del dron.

Como se puede apreciar en las imágenes, vemos lo que ve el dron, incluyendo sombras y texturas.

9.3 Pruebas con Láser 2D

Seguidamente, realizaremos pruebas con un láser 2D. Para ello integraremos un modelo con el plugin del láser y lo uniremos mediante un joint.

Para comprobar el funcionamiento del láser, lo pondremos entre 3 objetos. Estos objetos estarán a distintas distancias del láser.

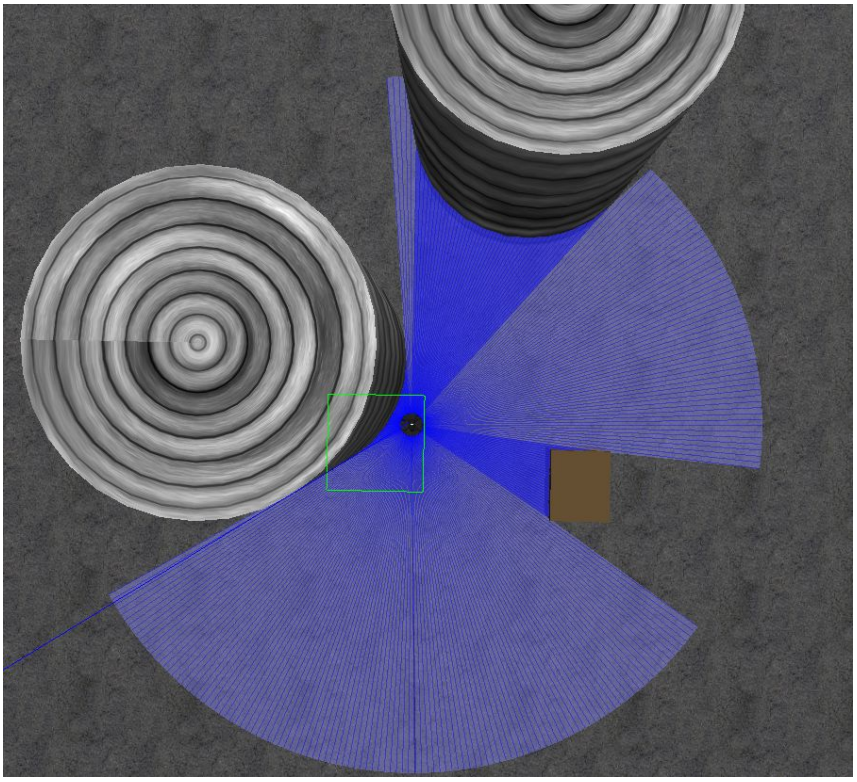

Figura 10.6: Imagen de la prueba en Gazebo.

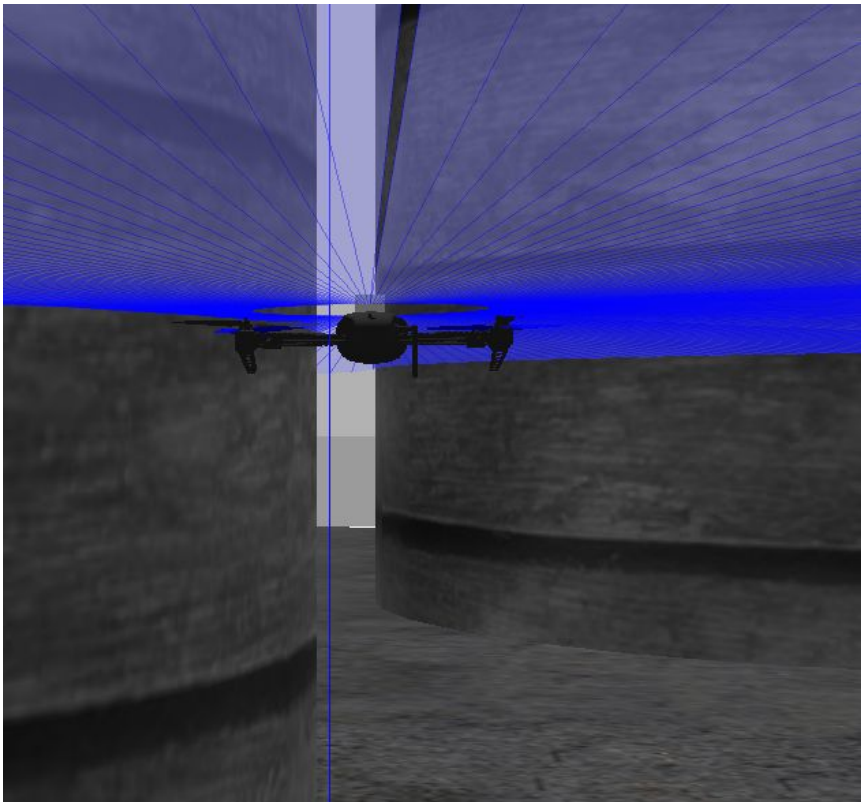

Figura 10.7: Dron con láser.

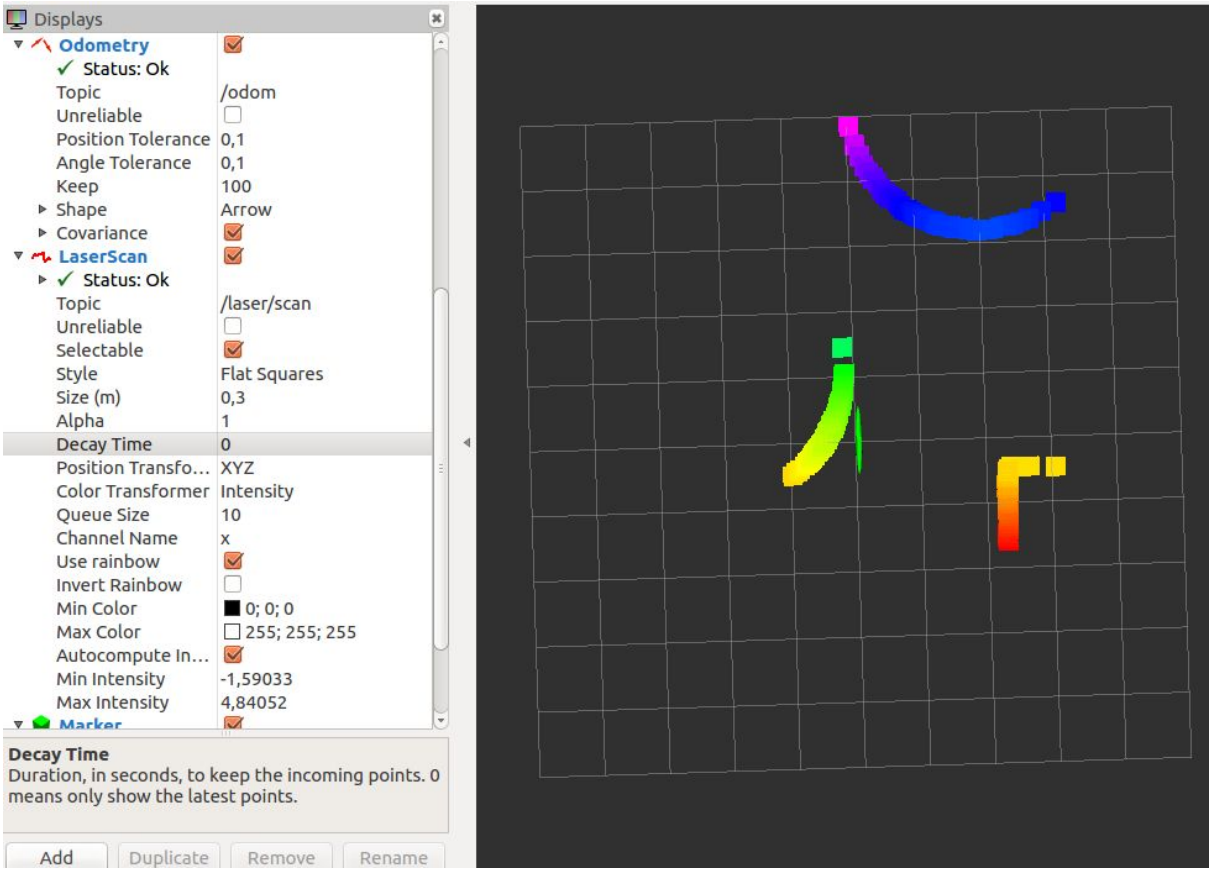

Figura 10.8: Visualización del láser en RVIZ.

Como podemos apreciar en la siguiente imagen, en RVIZ podemos visibilizar lo que el láser detecta. Poniendo como color transformer el valor intensity, podemos ver que dependiendo de la distancia a la cual está el objeto del láser, varía el color.

También podemos comprobar que los objetos que están cercanos al dron no se les puede identificar la forma tan facilmente.

11. Explicación Nodo ROS

En este apartado se muestra un nodo ROS programado durante el transcurso del proyecto. Durante esta explicación se divide el código en trozos, explicando paso por paso cada línea del código para así facilitar al lector el entendimiento del nodo.

También se detalla cómo construir un nodo ROS y como editar el fichero package.xml y el CMakelist.

11.1 Explicación del nodo

Este nodo actúa como mission planner básico. El objetivo es viajar a las coordenadas indicadas por el usuario en la terminal. Se crea una lista de misiones, se limpia la actual y se pasa la lista creada. Una vez la lista se ha transmitido al controlador, enviamos comandas para poner el dron en modo auto.mission y lo armamos.

Finalmente, una vez haber recorrido todos las coordenadas indicadas, hacemos que el dron se ponga en modo RTL, es decir, return to home. El home es una posición que el UAV reconoce.

11.2 Código

En este apartado trataremos las partes más importantes del nodo.

```
#include <ros/ros.h>
#include <mavros_msgs/CommandBool.h>
#include <mavros_msgs/SetMode.h>
#include <mavros_msgs/State.h>
#include <mavros_msgs/WaypointPull.h>
#include <mavros_msgs/WaypointPush.h>
#include <mavros_msgs/WaypointClear.h>
#include <mavros_msgs/WaypointReached.h>
#include <mavros_msgs/WaypointList.h>
#include <iostream>

using namespace std;
```

Incluimos los ficheros de cabecera importantes. Dado que vamos a limpiar y construir un listado de waypoints, necesitamos sus cabeceras. También necesitamos cambiar el estado del vehículo e iostream porque usaremos cout.

```
mavros_msgs::State current_state;
void state_cb(const mavros_msgs::State::ConstPtr& msg) {
    current_state = *msg;
}
```

Creación de una función callback. Este tipo de funciones son llamadas cuando un mensaje ha llegado al topic al cual está ligado.

```
int main (int argc, char **argv) {

     ros::init(argc,argv,"mission_planner");
     ros::NodeHandle nh;
```

Inicializamos un nodo ROS que tendrá como nombre mission_planner. Acto seguido declaramos un constructor de este nodo. Este elemento servirá para crear las suscripciones y las publicaciones.

```
 ros::Subscriber state_sub = nh.subscribe<mavros_msgs::State>
                ("mavros/state", 100, state_cb);
```

Nos suscribimos al tema /mavros/state, enlazándola a una llamada callback, la función state_cb anteriormente descrita. El segundo parámetro de esta función es el tamaño de la cola de los mensajes recibidos por el topic. Si llegamos a almacenar 100 mensajes porque no podemos procesar al mismo tiempo de publicación, se empezaran a desechar los mensajes más antiguos de la cola.

```
 ros::ServiceClient arming_client =
nh.serviceClient<mavros_msgs::CommandBool> ("mavros/cmd/arming");
```

Creamos un cliente para el topic mavros/cmd/arming. Un cliente nos permite utilizar servicios del topic, y en este caso, queremos poder armar el dron.

```
ros::Rate rate(20.0);
```

Declaramos la frecuencia a la cual queremos que los objetos ros funcionen.

```
if (wp_clear_client.call(wp_clear_srv))
            {
            ROS_INFO("Lista de waypoints limpiada");
            }
else
            {
            ROS_ERROR("Error en la limpieza de waypoints");
            }
```

Ejemplo de llamada al servicio. Estas llamadas son bloqueantes y no se seguirá ejecutando código hasta que se termine esta llamada. Si la llamada ha tenido exito, nos devolvera un valor true y si no ha tenido exito, nos devolvera false. En este caso estamos haciendo una llamada al servicio de WaypointClear, que borra la lista de misiones actuales del UAV.

```
for(int is = 0;is < num_waypoints;++is) {

            missionvector[is].command = 16;
            if(is == 0)
            missionvector[is].is_current = true;
            else
            missionvector[is].is_current = false;
            missionvector[is].autocontinue = true;
            missionvector[is].param1 = 0;
            missionvector[is].param2 = 0;
            missionvector[is].param3 = 0;
            missionvector[is].param4 = 0;
            float x;
            cout << "Añada X Coordenada" << endl;
            cin >> x;
            missionvector[is].x_lat =  x;
            float y;
            cout << "Añada la  Coordenada Y" << endl;
            cin >> y;
            missionvector[is].y_long = y;
            float z;
            cout << "Añada la Altura" << endl;
             cin >> z;
            missionvector[is].z_alt = z;
```

```
wp_push_srv.request.waypoints.push_back(missionvector[is]);

if (wp_srv_client.call(wp_push_srv))
```

Una vez limpiada la lista de misiones del dron, hemos de crear una nosotros. Una vez tenemos el número de misiones, se tiene que preguntar al usuario que coordenadas quiere para sus misiones. El valor iscurrent de Waypoint indica cual es la misión que se esta ejecutando actualmente. Obviamente al empezar la lista de 0, este será la primera mision. El parámetro autocontinue indica si al terminar esta misión pasamos al siguiente de la lista. El parámetro command indica el tipo de misión(en esta caso, todas serán misiones de pasar por la coordenada sin hacer nada). Los param1 no tienen valor con el valor de command actual.

Una vez hecha una misión, la enviamos al servicio y una vez ahí, la enviamos al UAV.

```
if(current_state.mode != "AUTO.MISSION") {
      ROS_INFO("AUTO SET");
      system("rosrun mavros mavsys mode -c AUTO.MISSION");
            }

if (!current_state.armed) {
      if( arming_client.call(arm_cmd) && arm_cmd.response.success)
{
            ROS_INFO("Vehicle armado");
            }
      }
```

Una vez enviadas las misiones, ponemos el dron en funcionamiento. Primero ponemos el dron en modo Mission, el cual hace que el dron haga las misiones. Acto seguido, lo armamos.

```
 while(ros::ok())  {
          if((current_waypoints.waypoints.size()-1) ==
reached_.wp_seq && current_state.mode == "AUTO.MISSION") {
                if(current_state.mode != "AUTO.RTL") {
                ROS_INFO("RETURN SET");
```

```
                system("rosrun mavros mavsys mode -c AUTO.RTL");
                                                }
                                }
        ros::spinOnce();
        rate.sleep();
              }
```

Finalmente, declaramos un while(ros::ok()). Esto provoca un bucle infinito salvo que se destruya el nodo. Esto puede ser provocado por:

- un SIGINT, es decir un Control + C.
- ros::shutdown()
- La destrucción de todos los nodos RosHandle.

En este bucle revisamos que si el dron ha alcanzado la última misión de la lista, vuelva inmediatamente a la posición home. Esto se efectúa llamando al servicio de estado y poniendo al dron en el modo RTL.

Finalmente, hacemos un spinOnce(), que es la llamada a las funciones callback. Sin esta función las funciones callback no funcionan. Finalmente ponemos el rate en sleep para que actúe con la frecuencia dada anteriormente.

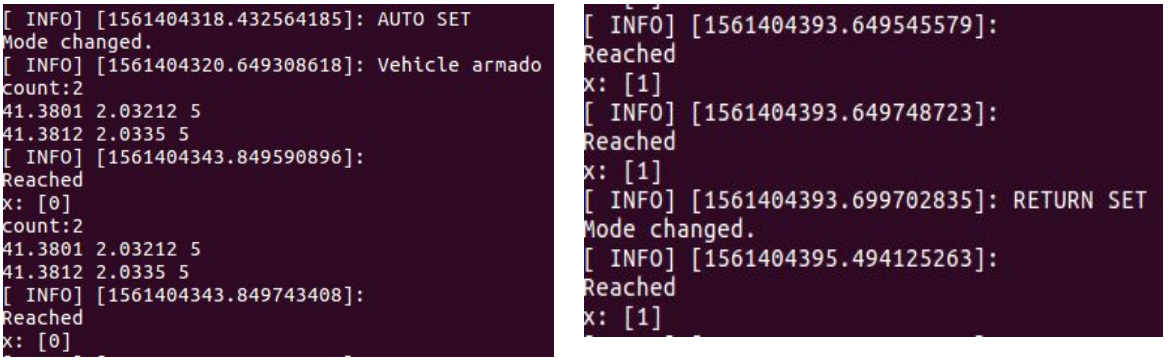

Figura 11.1: Salida de terminal de la ejecución del nodo.

12. Conclusión y líneas futuras.

En este apartado se detalla la conclusión del proyecto y la conclusión personal del desarrollador. También se mencionan posibles futuras mejoras para el proyecto.

12.1 Conclusión del proyecto

El objetivo de este proyecto era el de crear y proveer una herramienta que ayudará a testear futuros cambios en el modelaje o en el firmware. Para ello, se decidió crear un simulador HITL basado en la placa Pixhawk y en el firmware PX4.

Este simulador actualmente es capaz de:

- Ejecutar nodos ROS que afecten al comportamiento del UAV. Asimismo, también se puede visualizar los datos de este, datos como la IMU,GPS, etc.
- Implementar y visualizar sensores del UAV mediante herramientas ROS tales como RVIZ o rqt.
- Control Remoto mediante el mando Futaba T8J del UAV en el simulador.
- Creación de ficheros logs y posibilidad de análisis de estos ficheros mediante Flight Review.
- Configurar el UAV mediante parámetros en el QGC.
- Opción de ejecutar el simulador mediante la técnica SITL o la técnica HITL.
- Modelos propios de una EDAR. Asimismo, también se ha proveído de modelos básicos.

A nivel personal, creo que este proyecto me ha hecho aprender y robótica y me ha permitido entrar en el mundo la tecnología dron. Para efectuar este proyecto, he tenido que aprender diversas tecnologías como ROS y Gazebo, el funcionamiento de un drone, y de una controladora de vuelo. Estoy muy agradecido de haber podido entender todas estas tecnologías.

12.2 Líneas futuras del proyecto

Actualmente este simulador es capaz de testear el drone mediante un sistema de control remoto. Además, el simulador es capaz de utilizar e integrar nodos ROS y sensores en el sistema. Sin embargo, este proyecto tiene margen de mejora. A continuación, se ofrece un listado de posibles implementaciones para el sistema:

- **Actualizar modelos de Gazebo en tiempo de ejecución**: El simulador Gazebo nos permite añadir modelos en tiempo real de una forma simple. Sin embargo, el sistema no nos permite actualizar los modelos que están en el simulador, algo realmente importante. Por ejemplo, supongamos que tenemos un dron con una botella de medio litro que recoge agua. Durante el trayecto de ida, esta aeronave pesa X kilos, pero en el trayecto de vuelta pesa X + 0.5 kilos y este peso extra puede afectar al dron, además de la visibilidad del cambio de estado. Por eso es necesario crear plugins que permitan cambios en los modelos mediante el uso de la API de Gazebo.
- **Construcción de un PID más robusto**: El PID proporcionado por PX4 es poco robusto para el uso que se le quiere dar a la plataforma UAV, sería necesario refinar su calibración para darle más robustez el sustituir la cascada PID por un control más robusto y tolerante a perturbaciones.
- **Modelado personalizado:** el quadcopter del departamento está destruido y necesitará cambios. Una vez se hayan efectuado estos cambios, se debería cambiar las características del fichero iris.sdf para poner las de este nuevo dron. Además, se deberían crear ficheros .dae o .stl para cambiar la visualización del quadcopter.
- **Añadir microprocesador al sistema:** Hay 2 motivos por el cual es ideal conectar un microprocesador al sistema. El primero es que en la realidad, el dron no estará conectado al PC, y por lo tanto no podrá ejecutar nodos ROS. El segundo motivo es la sobrecarga del PC. Actualmente el sistema ejecuta todo en el ordenador(Gazebo, QGC, Nodos ROS, conexiones, etc.) Es posible que al ejecutar un nodo ROS con alta carga computacional, el rendimiento del sistema baje considerablemente. Por lo tanto, tener un microprocesador para ejecutar los nodos ROS es ideal.

13. Bibliografía

Referencias

[1]Hardware-in-the-Loop. Wikipedia [en línea]. [Consultada: 25 Febrero 2019]. Recuperado de: < https://es.wikipedia.org/wiki/Hardware-in-the-loop >

[2] Gazebo (2019). Gazebo [en línea]. [Consultada: 23 Febrero 2019]. Recuperado de: < http://gazebosim.org/ >

[3] Documentation ROS. Ros[en línea]. [Consultada: 22 Febrero 2019]. Recuperado de: < http://wiki.ros.org/ROS/ >

[4] Trump amenaza a Irán tras el derribo de un dron estadounidense en Ormuz(21 de junio de 2019) . El País [en línea]. [Consultada: 24 Junio 2019]. Recuperado de: <https://elpais.com/internacional/2019/06/20/actualidad/1561005154_875538.html >

[5] Origen y desarrollo de los drones. Drones.uv[en línea]. [Consultada: 20 Junio 2019]. Recuperado de: < http://drones.uv.es/origen-y-desarrollo-de-los-drones/ >

[6] Ryan Model 124 / BQM-34A Firebee . Wikipedia [en línea]. [Consultada: 21 Junio 2019]. Recuperado de: <https://en.wikipedia.org/wiki/Ryan_Firebee >

[7] Cuál va a ser el futuro de los drones (21 Noviembre 2017). Altran[en línea]. [Consultada: 22 Febrero 2019]. Recuperado de: <https://equipo.altran.es/cual-va-a-ser-el-futuro-de-los-drones/ >

[8] Peter Lepej, Peter Santamaria, Joan Angel Sola, Lamia Chaari (2019). UPC.A flexible hardware-in-the-loop architecture for UAVs [en línea]. [Consultada: 31 Enero 2019]. Recuperado de: <https://discovery.upc.edu/iii/encore/record/C__Rx1184374__SA%20flexible%20hardware-in-the-loop__Orightresult__U__X4?lang=cat&suite=def >

[9] Ha, Nyugen, Jahn (2019). Universidad de Ulsan, Corea del Sur. Development of a new hybrid drone and software-in-the-loop simulation using PX4 code [en línea]. [Consultada: 24 Febrero 2019]. Recuperado de: < http://eds.b.ebscohost.com.recursos.biblioteca.upc.edu/eds/detail/detail?vid=0&sid=fa5c8145-f2db-476f-8401-6c4b94e22457%40sessionmgr103&bdata=Jmxhbmc9ZXMmc2l0ZT1lZHMtbGl2ZQ%3d%3d >

[10] Seung-hyeon Cheon (2019). Gyeongsang National University, Corea del Sur. Hardware-In-the-Loop Simulation Platform for Image-based Object Tracking Method using Small UAV [en línea]. [Consultada: 24 Febrero 2019].
Recuperado de: < https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/stamp/stamp.jsp?tp=&arnumber=7778031 >

[11] Mavros. Wiki Ros [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: < http://wiki.ros.org/mavros >

[12] MAVLink. Wikipedia [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: < https://en.wikipedia.org/wiki/MAVLink >

[13] Bernhard Fuerfanger (2019). UPC.Modelling and control of a customized UAV with gimbal-attachment [en línea]. [Consultada: 18 Marzo 2019].
Recuperado de:
<https://discovery.upc.edu/iii/encore/record/C__Rb1516817__Smodelling%20and%20control%20of%20a%20customized__Orightresult__U__X6?lang=cat&suite=def >

[14] Desarrollo en Cascada. Wikipedia [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: < https://es.wikipedia.org/wiki/Desarrollo_en_cascada >

[15] Git. Git [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: <https://git-scm.com>

[16] Pixhawk. Pixhawk [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: <http://pixhawk.org/>

[17] Google Drive. Google Drive [en línea]. [Consultada: 23 Marzo 2019].
Recuperado de: <https://www.google.com/intl/es_ALL/drive/>

[18] Gantter. Gantter [en línea]. [Consultada: 1 Marzo 2019].
Recuperado de: <https://www.gantter.com/>

[19] Trello. Trello [en línea]. [Consultada: 29 Marzo 2019].
Recuperado de: <http:/trello.com/>

[20] L. Meier, D. Honegger, and M. Pollefeys, “Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms,” in2015 IEEE International Conference on Robotics and Automation (ICRA), 2015.
[Consultada: 15 Febrero 2019]. Recuperado de
<https://www.academia.edu/28629803/PX4_A_Node-Based_Multithreaded_Open_Source_Robotics_Framework_for_Deeply_Embedded_Platforms >

[21] Futaba T8J. FutabaArc [en línea]. [Consultada: 24 Marzo 2019].
Recuperado de: <https://www.futabarc.com/systems/futk8100-8j/index.html >

[22] Silicon Errata. Docs PX4 [en línea]. [Consultada: 26 Abril 2019].
Recuperado de: <https://docs.px4.io/en/flight_controller/silicon_errata.html >

[23] Mavlink-Router .Github [en línea]. [Consultada: 28 Abril 2019].
Recuperado de: <https://github.com/intel/mavlink-router >

[24] PagePersonal(2019). Tendencias del Mundo Laboral, Tecnología. [Consultada: 23 Febrero 2019]. Recuperado de: <
https://www.pagepersonnel.es/sites/pagepersonnel.es/files/PG_ER_IT_2018.pdf >

[25] Gazebo + ROS. Gazebo [en línea]. [Consultada: 3 Junio 2019].
Recuperado de: < http://gazebosim.org/tutorials?tut=ros_overview >

[26] tf. Wiki Ros [en línea]. [Consultada: 4 Junio 2019].
Recuperado de: < http://wiki.ros.org/tf >

[27] rviz. Wiki Ros [en línea]. [Consultada: 4 Junio 2019].
Recuperado de: < http:/wiki.ros.org/rviz >

[28] Aicha Driss, Lobna Krichen, Fourati Mohamed, Lamia Chaari (2019). Universidad de SFax, Túnez. Simulation Tools, Environments and Frameworks for UAV Systems Performance Analysis [en línea]. [Consultada: 24 Febrero 2019].
Recuperado de:
<https://www.researchgate.net/publication/327323613_Simulation_Tools_Environments_and_Frameworks_for_UAV_Systems_Performance_Analysis >

[29] rotors_simulators. Wiki Ros [en línea]. [Consultada: 23 Febrero 2019]. Recuperado de: < http://wiki.ros.org/rotors_simulator >

[30] X-Plane. X-Plane [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: < https://www.x-plane.com/ >

[31] JMAVSim .Github [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: <https://github.com/PX4/jMAVSim >

[32] Airsim. Github [en línea]. [Consultada: 23 Febrero 2019].
Recuperado de: <https://github.com/microsoft/AirSim >

[33] Pixhack 2.8.4 .Rcgroups [en línea]. [Consultada: 10 Junio 2019].
Recuperado de:
<https://www.rcgroups.com/forums/showthread.php?2707268-Pixhack-2-8-4-32-bit-Flight-Controller-Based-on-Pixhawk-Autopilot >

[34] SDF. SDF [en línea]. [Consultada: 23 Mayo 2019].
Recuperado de: <http://sdformat.org/ >

[35] URDF. WikiRos [en línea]. [Consultada: 23 Mayo 2019].
Recuperado de: <http://wiki.ros.org/urdf >

[36] Model kinematics. SDF [en línea]. [Consultada: 20 Junio 2019].
Recuperado de:
<http://sdformat.org/tutorials?tut=spec_model_kinematics&cat=specification& >

[37] EDAR Sant Feliu. Google Maps [en línea]. [Consultada: 23Junio 2019].
Recuperado de:
<https://www.google.com/maps/place/Edar+St+Feliu/@41.3773727,2.03227,362a,35y,39.4t/data=!3m1!1e3!4m5!3m4!1s0x12a49bab87328f51:0xd4fec12372dfe540!8m2!3d41.3804911!4d2.0321049 >

Anexos

Anexo A

A.1 Instalación del simulador

Estos son los requisitos mínimos para poder ejecutar el simulador.

- Procesador I5
- Ubuntu 16.04
- Tarjeta gráfica de 64 bits.

A continuación se ofrecen los scripts para la instalación del simulador. Esto incluye la instalación de ROS Kinetic, Gazebo 7, creación de un entorno catkin, mavros y mavlink.

A.1.1 Script ubuntu_sim_ros_gazebo.sh

```
#!/bin/bash

## Bash script for setting up a ROS/Gazebo development environment
for PX4 on Ubuntu LTS (16.04).
## It installs the common dependencies for all targets (including
Qt Creator) and the ROS Kinetic/Gazebo 7 (the default).
##
## Installs:
## - Common dependencies libraries and tools as defined in
`ubuntu_sim_common_deps.sh`
## - ROS Kinetic (including Gazebo7)
## - MAVROS

echo "Downloading dependent script 'ubuntu_sim_common_deps.sh'"
# Source the ubuntu_sim_common_deps.sh script directly from github
common_deps=$(wget
https://raw.githubusercontent.com/PX4/Devguide/master/build_script
s/ubuntu_sim_common_deps.sh -O -)
wget_return_code=$?
# If there was an error downloading the dependent script, we must
warn the user and exit at this point.
if [[ $wget_return_code -ne 0 ]]; then echo "Error downloading
'ubuntu_sim_common_deps.sh'. Sorry but I cannot proceed further
:("; exit 1; fi
```

```
# Otherwise source the downloaded script.
. <(echo "${common_deps}")

# ROS Kinetic/Gazebo (ROS Kinetic includes Gazebo7 by default)
## Gazebo simulator dependencies
sudo apt-get install protobuf-compiler libeigen3-dev libopencv-dev
-y

## ROS Gazebo: http://wiki.ros.org/kinetic/Installation/Ubuntu
## Setup keys
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu
$(lsb_release -sc) main" >
/etc/apt/sources.list.d/ros-latest.list'
sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80
--recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
## For keyserver connection problems substitute
hkp://pgp.mit.edu:80 or hkp://keyserver.ubuntu.com:80 above.
sudo apt-get update
## Get ROS/Gazebo
sudo apt-get install ros-kinetic-desktop-full -y
## Initialize rosdep
sudo rosdep init
rosdep update
## Setup environment variables
rossource="source /opt/ros/kinetic/setup.bash"
if grep -Fxq "$rossource" ~/.bashrc; then echo ROS setup.bash
already in .bashrc;
else echo "$rossource" >> ~/.bashrc; fi
eval $rossource
## Get rosinstall
sudo apt-get install python-rosinstall -y

# MAVROS: https://dev.px4.io/en/ros/mavros_installation.html
## Create catkin workspace
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws

## Install dependencies
sudo apt-get install python-wstool python-rosinstall-generator
python-catkin-tools -y
```

```
## Initialise wstool
wstool init ~/catkin_ws/src

## Build MAVROS
### Get source (upstream - released)
rosinstall_generator --upstream mavros | tee
/tmp/mavros.rosinstall
### Get latest released mavlink package
rosinstall_generator mavlink | tee -a /tmp/mavros.rosinstall
### Setup workspace & install deps
wstool merge -t src /tmp/mavros.rosinstall
wstool update -t src
if ! rosdep install --from-paths src --ignore-src --rosdistro
kinetic -y; then
    # (Use echo to trim leading/trailing whitespaces from the
unsupported OS name
    unsupported_os=$(echo $(rosdep db 2>&1| grep Unsupported | awk
-F: '{print $2}'))
    rosdep install --from-paths src --ignore-src --rosdistro
kinetic -y --os ubuntu:xenial
fi
## Build!
catkin build
## Re-source environment to reflect new packages/build environment
catkin_ws_source="source ~/catkin_ws/devel/setup.bash"
if grep -Fxq "$catkin_ws_source" ~/.bashrc; then echo ROS
catkin_ws setup.bash already in .bashrc;
else echo "$catkin_ws_source" >> ~/.bashrc; fi
eval $catkin_ws_source

echo "Downloading dependent script
'install_geographiclib_datasets.sh'"
# Source the install_geographiclib_datasets.sh script directly
from github
install_geo=$(wget
https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scr
ipts/install_geographiclib_datasets.sh -O -)
wget_return_code=$?
# If there was an error downloading the dependent script, we must
warn the user and exit at this point.
if [[ $wget_return_code -ne 0 ]]; then echo "Error downloading
```

```
'install_geographiclib_datasets.sh'. Sorry but I cannot proceed
further :("; exit 1; fi
# Otherwise source the downloaded script.
sudo bash -c "$install_geo"

# Common dependencies
echo "Installing common dependencies"
sudo apt-get update -y
sudo apt-get install git zip qtcreator cmake build-essential
genromfs ninja-build exiftool -y
# Required python packages
sudo apt-get install python-argparse python-empy python-toml
python-numpy python-dev python-pip -y
sudo -H pip install --upgrade pip
sudo -H pip install pandas jinja2 pyserial
# optional python tools
sudo -H pip install pyulog

# Clone PX4/Firmware
clone_dir=~/src
echo "Cloning PX4 to: $clone_dir."
if [ -d "$clone_dir" ]
then
    echo " Firmware already cloned."
else
    mkdir -p $clone_dir
    cd $clone_dir
    git clone https://github.com/PX4/Firmware.git
fi

if [[ ! -z $unsupported_os ]]; then
    >&2 echo -e "\033[31mYour OS ($unsupported_os) is unsupported.
Assumed an Ubuntu 16.04 installation,"
    >&2 echo -e "and continued with the installation, but if
things are not working as"
    >&2 echo -e "expected you have been warned."
fi
```

Anexo B: Código de los sensores y modelos.

En Github se encuentra todo lo utilizado para la simulación en el siguiente enlace:https://github.com/Manec/PX4. Los nodos ros creados se encuentran en: https://github.com/Manec/ROS

A continuación se muestra el código usado en las pruebas de la memoria.

Anexo B.1 Camara fpv_cam

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="fpv_cam">
    <pose>0 0 0.036 0 0 0</pose>
    <link name="link">
       <inertial>
          <mass>0.015</mass>
          <inertia>
            <ixx>4.15e-6</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>2.407e-6</iyy>
          <iyz>0</iyz>
          <izz>2.407e-6</izz>
          </inertia>
        </inertial>
        <collision name='collision'>
          <geometry>
            <box>
              <size>0.01 0.01 0.01</size>
            </box>
          </geometry>
          <max_contacts>10</max_contacts>
          <surface>
            <contact>
              <ode/>
            </contact>
            <bounce/>
            <friction>
              <ode/>
            </friction>
```

```
          </surface>
        </collision>
        <visual name='visual'>
          <geometry>
            <box>
              <size>0.01 0.01 0.01</size>
            </box>
          </geometry>
        </visual>
        <sensor name='camera' type='camera'> <!-- Cambiar tipo por
Depth en caso de querer camara depth -->


          <camera name='__default__'>
            <horizontal_fov>1.047</horizontal_fov>
            <image>
              <width>480</width>
              <height>320</height>
            </image>
            <clip>
              <near>0.1</near>
              <far>100</far>
            </clip>
            <lens>
              <type>custom</type>
              <custom_function>
                <c1>1.05</c1>
                <c2>4</c2>
                <f>1</f>
                <fun>tan</fun>
              </custom_function>
              <scale_to_hfov>1</scale_to_hfov>
              <cutoff_angle>3.1416</cutoff_angle>
              <env_texture_size>1080</env_texture_size>
            </lens>
          </camera>
          <always_on>1</always_on>
          <update_rate>30</update_rate>
          <visualize>1</visualize>
          <plugin name='camera_plugin'
filename='libgazebo_ros_camera.so'>
```

```
<!-- Cambiar plugin por libgazebo_ros_openni_kinect en caso de
camara depth -->
            <alwaysOn>true</alwaysOn>
            <imageTopicName>image_raw</imageTopicName>
            <cameraInfoTopicName>camera_info</cameraInfoTopicName>
            <updateRate>30.0</updateRate>
            <cameraName>usb_cam</cameraName>
            <frameName>/base_camera_link</frameName>
            <CxPrime>320.5</CxPrime>
            <Cx>480.5</Cx>
            <Cy>320.5</Cy>
            <hackBaseline>0</hackBaseline>
            <focalLength>277.191356</focalLength>
            <distortionK1>0.0</distortionK1>
            <distortionK2>0.0</distortionK2>
            <distortionK3>0.0</distortionK3>
            <distortionT1>0.0</distortionT1>
            <distortionT2>0.0</distortionT2>
          </plugin>
        </sensor>
        <self_collide>0</self_collide>
        <kinematic>0</kinematic>
      </link>
  </model>
</sdf>
```

B.2 Código .sdf del Rangefinder

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <model name="lidar">
    <link name="link">
      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <mass>0.01</mass>
        <inertia>
          <ixx>2.1733e-6</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>2.1733e-6</iyy>
```

```
          <iyz>0</iyz>
          <izz>1.8e-7</izz>
        </inertia>
      </inertial>
      <visual name="visual">
        <geometry>

<cylinder><radius>0.006</radius><length>0.05</length></cylinder>
        </geometry>
        <material>
          <script>
            <name>Gazebo/Black</name>
          </script>
        </material>
      </visual>
      <sensor name="laser" type="ray">
        <pose>0 0 0 0 -1.570896 0</pose>
        <ray>
          <scan>
            <horizontal>
              <samples>1</samples>
              <resolution>1</resolution>
              <min_angle>-0</min_angle>
              <max_angle>0</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.06</min> <!-- Retocar dependiendo del joint -->
            <max>35</max>
            <resolution>0.01</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <mean>0.0</mean>
            <stddev>0.02</stddev>
          </noise>
        </ray>
        <plugin name="LaserPlugin"
filename="libgazebo_lidar_plugin.so">
          <robotNamespace>Rangefinder</robotNamespace>
          <min_distance>0.2</min_distance> <!-- Retocar
```

```
dependiendo del joint -->
          <max_distance>15.0</max_distance>
        </plugin>
        <always_on>1</always_on>
        <update_rate>20</update_rate>
        <visualize>true</visualize>
      </sensor>
    </link>
  </model>
</sdf>
```

B.3 Código .sdf del láser 2D

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="rplidar">
    <link name="link">

      <inertial>
        <pose>0 0 0 0 0 0</pose>
        <mass>0.19</mass>
        <inertia>
          <ixx>4.15e-6</ixx>
          <ixy>0</ixy>
          <ixz>0</ixz>
          <iyy>2.407e-6</iyy>
          <iyz>0</iyz>
          <izz>2.407e-6</izz>
        </inertia>
      </inertial>

      <visual name="visual">
        <geometry>
          <box>
            <size>0.02 0.05 0.05</size>
          </box>
        </geometry>
      </visual>

      <sensor name="laser" type="gpu_ray">
        <ray>
```

```
          <scan>
            <horizontal>
              <samples>360</samples>
              <resolution>1</resolution>
              <min_angle>-3.14</min_angle>
              <max_angle>3.14</max_angle>
            </horizontal>
          </scan>
          <range>
            <min>0.2</min>
            <max>6</max>
            <resolution>0.05</resolution>
          </range>
          <noise>
            <type>gaussian</type>
            <mean>0.0</mean>
            <stddev>0.01</stddev>
          </noise>
        </ray>
        <plugin name="laser" filename="libGpuRayPlugin.so" />
        <plugin name="gazebo_ros_head_rplidar_controller"
filename="libgazebo_ros_gpu_laser.so">
          <topicName>laser/scan</topicName>
          <frameName>base_link</frameName>
        </plugin>
        <always_on>1</always_on>
        <update_rate>5.5</update_rate>
        <visualize>true</visualize>
      </sensor>
    </link>
  </model>
</sdf>
```

B.4 Xacro de Camara y Laser Hokuyo

```
<!-- Front Laser -->
<xacro:macro name="laser" params="name parent xyz rpy" >

  <joint name="${parent}_${name}_joint" type="fixed">
      <axis xyz="0 0 1" />
      <origin xyz="${xyz}" rpy="${rpy}"/>
```

```
      <parent link="${parent}"/>
      <child link="${name}"/>
    </joint>

    <link name="${name}">
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
          <box size="0.1 0.1 0.1"/>
        </geometry>
      </collision>
      <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
           <box size="0.1 0.1 0.1"/>
        </geometry>
      </visual>
      <inertial>
        <mass value="1e-5" />
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0"
izz="1e-6" />
      </inertial>
    </link>

    <gazebo reference="${name}">
        <sensor name="laser" type="ray">
        <pose>0 0 0 0 0 0</pose>
          <ray>
            <scan>
              <horizontal>
                <!-- The URG-04LX-UG01  has  683 steps with
0.35139 Degree resolution -->
                <resolution>1</resolution>
                <max_angle>2.0944</max_angle> <!-- 120 Degree -->
                <min_angle>-2.0944</min_angle> <!-- -120 Degree
-->
                <samples>683</samples>
              </horizontal>
            </scan>
            <range>
```

```
            <min>0.08</min>
            <max>5</max>
            <resolution>0.01</resolution>
          </range>
        </ray>

        <plugin name="laser" filename="libgazebo_ros_laser.so" >
          <robotNamespace></robotNamespace>
          <topicName>${name}/scan</topicName>
          <frameName>${name}</frameName>
        </plugin>
        <!--
        <plugin name="laser" filename="libRayPlugin.so" />
        -->

        <always_on>1</always_on>
        <update_rate>30</update_rate>
        <visualize>true</visualize>
      </sensor>
    </gazebo>
</xacro:macro>


  <!-- Macro to add a camera. -->
  <xacro:macro name="camera_macro"
    params="namespace parent_link camera_suffix frame_rate
      horizontal_fov image_width image_height image_format
min_distance
      max_distance noise_mean noise_stddev enable_visual *cylinder
*origin">
    <link name="${namespace}/camera_${camera_suffix}_link">
      <collision>
        <origin xyz="0 0 0" rpy="0 0 0" />
        <geometry>
          <xacro:insert_block name="cylinder" />
        </geometry>
      </collision>
      <xacro:if value="${enable_visual}">
        <visual>
          <origin xyz="0 0 0" rpy="0 1.57079632679 0" />
          <geometry>
```

```
            <xacro:insert_block name="cylinder" />
          </geometry>
          <material name="red" />
        </visual>
      </xacro:if>
      <inertial>
        <mass value="1e-5" />
        <origin xyz="0 0 0" rpy="0 0 0" />
        <inertia ixx="1e-6" ixy="0" ixz="0" iyy="1e-6" iyz="0"
izz="1e-6" />
      </inertial>
    </link>
    <joint name="${namespace}/camera_${camera_suffix}_joint"
type="fixed">
      <xacro:insert_block name="origin" />
      <parent link="${parent_link}" />
      <child link="${namespace}/camera_${camera_suffix}_link" />
    </joint>
    <gazebo reference="${namespace}/camera_${camera_suffix}_link">
      <sensor type="camera"
name="${namespace}_camera_${camera_suffix}">
        <update_rate>${frame_rate}</update_rate>
        <camera name="head">
          <horizontal_fov>${horizontal_fov}</horizontal_fov>
          <image>
            <width>${image_width}</width>
            <height>${image_height}</height>
            <format>${image_format}</format>
          </image>
          <clip>
            <near>${min_distance}</near>
            <far>${max_distance}</far>
          </clip>
          <noise>
            <type>gaussian</type>
            <!-- Noise is sampled independently per pixel on each
frame.
                 That pixel's noise value is added to each of its
color
                 channels, which at that point lie in the range
[0,1]. -->
```

```
            <mean>${noise_mean}</mean>
            <stddev>${noise_stddev}</stddev>
          </noise>
        </camera>
        <plugin
name="${namespace}_camera_${camera_suffix}_controller"
filename="libgazebo_ros_camera.so">
          <robotNamespace>${namespace}</robotNamespace>
          <alwaysOn>true</alwaysOn>
          <updateRate>${frame_rate}</updateRate>
          <cameraName>camera_${camera_suffix}</cameraName>
          <imageTopicName>image_raw</imageTopicName>
          <cameraInfoTopicName>camera_info</cameraInfoTopicName>
          <frameName>camera_${camera_suffix}_link</frameName>
          <hackBaseline>0.0</hackBaseline>
          <distortionK1>0.0</distortionK1>
          <distortionK2>0.0</distortionK2>
          <distortionK3>0.0</distortionK3>
          <distortionT1>0.0</distortionT1>
          <distortionT2>0.0</distortionT2>
        </plugin>
      </sensor>
    </gazebo>
  </xacro:macro>
```